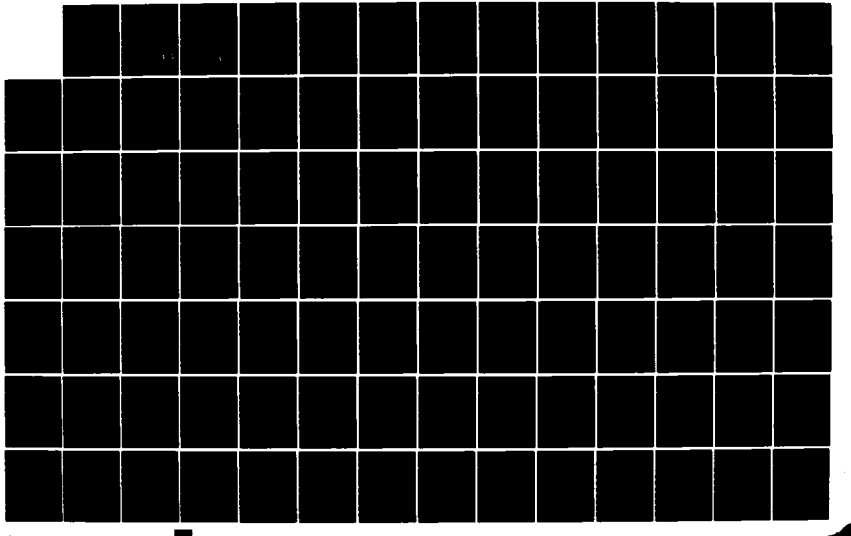
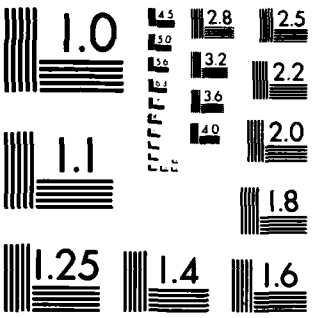
AD-A151 935 CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE UNIVERSAL NETWORK INTERFA.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C T CHILDRESS 1/2

UNCLASSIFIED DEC 84 AFIT/GE/ENG/84D-17-VOL-1 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963 A

DTIC

(1)

NETWORK DEVELOPMENT AND IMPLEMENTATION OF THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID) IN THE DIGITAL ENGINEERING LABORATORY NETWORK (DELNET)

VOL I OF II

THESIS

AFIT/GE/ENG/84D-17 Creed E. Childress, Jr.
Captain USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/84D-17

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE COAXIAL NETWORK INTERFACE DEVICE (CNID) II IN THE DIGITAL ENGINEERING LABORATORY NETWORK (DELNET) VOL I OF II

THESIS

AFIT/GE/ENG/84D-17 Creed T. Childress, Jr.
Captain USAF

AFIT/GE/ENG/84D-17

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II IN THE DIGITAL ENGINEERING LABORATORY NETWORK (DELNET) VOL I OF II

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Creed T. Childress, Jr., BS EE, BS IOE

Captain, USAF

December 1984

PREFACE

This research effort describes the continued development of an Improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two hardware modules: a local module for the network layer software and a network module for the data link layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an iSBC 544 and the network module is an iSBC 88/45. This report documents the detailed hardware and software design, test, and integration of this system.

I especially thank my thesis advisor, Dr. Gary Lamont, for his persistence and perseverance in nudging me through this challenging and successful endeavor. My thanks as well to Major Walter Seward and Major Kenneth Castor for their valuable comments, suggestions, and encouragement during the course of this investigation. I greatly appreciate the assistance from Mr. Orville Wright, Mr. Charlie Powers, Capt Lee Baker, and Mr. Robert Durham for their technical and supply support. To Capt Ken Cole goes special appreciation for the stimulating and challenging theoretical, technical and recreational conversations in the UNID developmental phases during the dead of the night. Last, but certainly not least, to my two sons, Marek and Derrick: We will soon have the time for the fishing, skiing, hiking and hunting that we have foregone these last 18 months; thank you for your patience. To my wife Regina goes special appreciation for her patience and understanding.

Table of Contents

Table of Contents (cont)

Table of Contents (cont)

Table of Contents (cont)

List of Figures

List of Figures (cont)

List of Tables

Abstract

This research effort describes the continued development of an Improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two main hardware modules; a local module for the network layer software and a network module for the data link layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an Intel iSBC 544 and the network module is an Intel iSBC 86/45. The network layer software supports the CCITT X.25, datagram option, protocol and the data link layer software supports the CCITT X.25 LAPB (HDLC) protocol. This report documents the detailed hardware and software design, integration, validation and test of this system.

CONTINUED DEVELOPMENT AND IMPLEMENTATION
OF THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
IN THE
DIGITAL ENGINEERING LABORATORY NETWORK (DELNET)

I. Introduction and Background

Introduction

This thesis describes the development and implementation of the Universal Network Interface Device (UNID) II. Chapter One provides an introduction to the UNID and the Digital Engineering Laboratory Network (DELNET). Chapter Two elaborates on the requirements and standards used in the UNID and DELNET. Chapter Three explains the hardware architecture and Chapter Four elaborates upon the supporting software design. Chapter Five describes the testing philosophy, design and results while Chapter Six concludes the thesis with a summary and recommendations for future work.

Chapter One begins with the background of the Digital Engineering Laboratory Network (DELNET) and the Universal Network Interface Device (UNID) I. It then covers the current status of the UNID II, problem statement, scope, assumptions, standards, approach, equipment, and other support required to develop and implement a working UNID II.

Background

In 1977, a report from the 1342nd Electronics Engineering Group (AFCC) stated that a local area network (LAN) would be an ideal candidate to connect the myriad of electronic data devices resident on a typical Air Force Base (1). The authors of the report saw the need for

some method of connecting these equipments in such a way so that user data processing and telecommunications needs could be efficiently and effectively satisfied. The report summarized that a multi-ring network concept would be a primary candidate for a typical base level teleprocessing and narrative message network of the late 1980's. A key to the multi-ring concept was the development of five different functional devices to handle the requirements of interfacing the diverse user equipment to the multiple ring structure. This basic idea was expanded and tasked to Rome Air Development Center (RADC) for incorporation into a post doctoral study program (75). An initial concept of the multi-ring, base level network is shown in Figure 1-1.

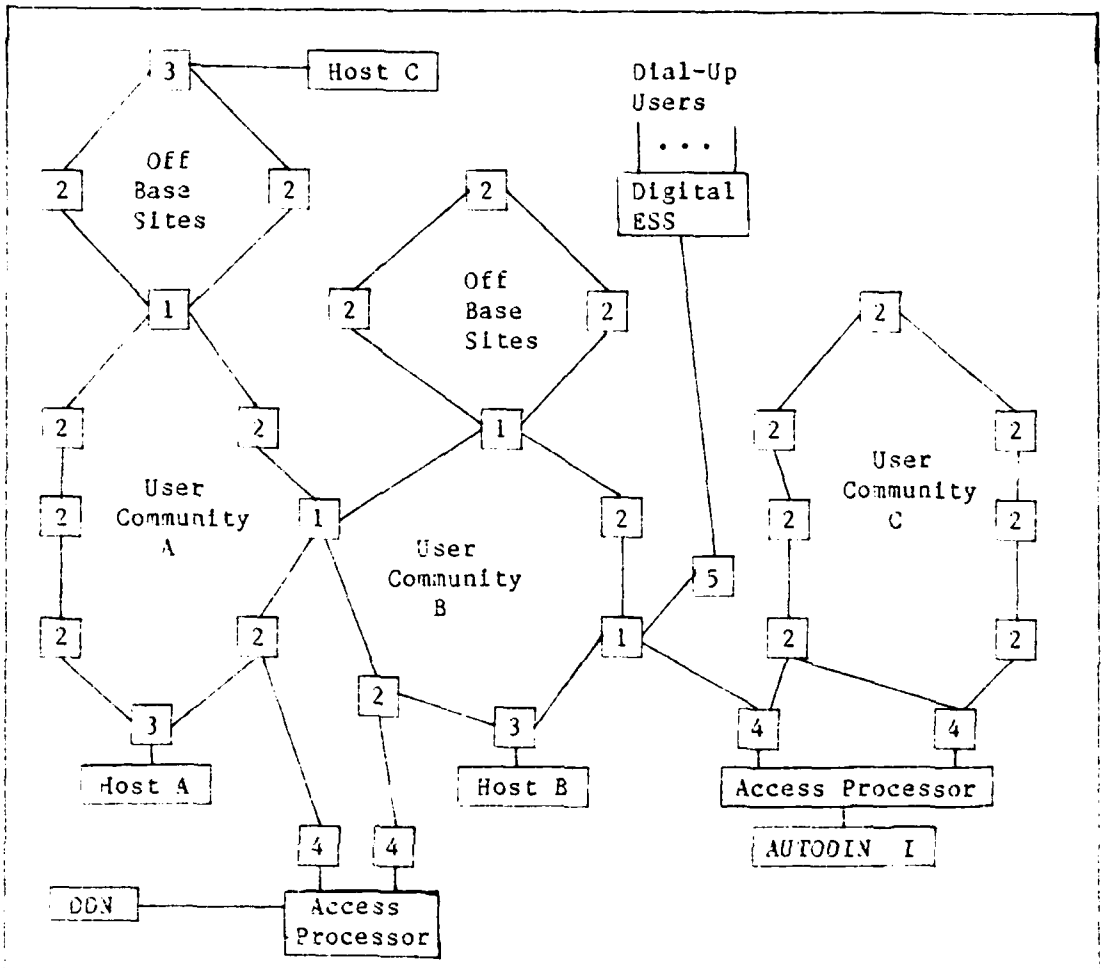


Figure 1-1. Initial Concept of a Multi-Ring Base Level Network (75)

Further investigation of the system requirements and functional definitions revealed that the five different devices could be incorporated into one piece of hardware. Since the single device must interface many different user equipments with the multiple ring structure, it was given the name Universal Network Interface Device, or UNID, for short.

It was at this point that the Air Force Institute of Technology (AFIT) became involved. An overall plan was developed for the architectural, hardware and software designs of both the DELNET and the UNID. The plan also included the use of the DELNET and UNID for educational purposes in computer network courses and research. Master's level thesis efforts began in 1978 with Sluzevich who outlined the initial conceptual hardware and software designs for the UNID. Sluzevich separated the design of the UNID into four tasks (84):

1. Define the functional requirements of the UNID.
2. Translate functional requirements into system design.
3. Design the UNID's hardware.
4. Design the UNID's software.

The UNID began its evolution in a modular design which would allow the required degree of universality. The design consisted of three logical parts. The first part interfaced the UNID with the host equipment. The second interfaced the UNID with the multiple ring LAN. The third part was a processor to provide the control and processing capabilities for the first and second pieces. In 1979, Brown (7) upgraded the basic design by expanding the processor into two processors, one for the local host equipment and the second for the ring LAN. A shared block of memory was also included in the design to allow the two processors to move data from the local to network hardware and vice versa. In 1980, Baker (4) addressed the need for further software development and testing of both the hardware and software required to obtain an operational device. He also improved existing hardware and software development tools to accomplish the UNID hardware and software testing. In 1981, Papp (74) developed the operational hardware for the

UNID. His work also included complete documentation and testing. Cuomo (11), in 1982, improved the hardware design by changing the processor memory from dynamic to static, including four ports for the local host processors, and improved the internal wiring to minimize noise and spurious interference. Spear (86), in 1983, began a through redesign and implementation of the UNID I hardware to eliminate many timing and other wiring problems that had heretofore existed.

At the same time Sluzevich began the UNID development, Ravenscroft began work on the conceptual design of a local area network for the Digital Engineering Laboratory (DEL), later called the DELNET (78). Hobart (33), in early 1981, began the conceptual development of the software which would eventually be needed for the DEL. He used structured analysis and design techniques (SADTs) to develop the initial data flow diagrams (DFD) for the network. In 1981, Geist (29) began the protocol development for the UNID. He used the International Standards Organization (ISO) Open Systems Interconnection (OSI) (12) model as the basic framework for software design and development in an effort to provide a standardized set of network protocols. He further refined the initial data flow requirements and documentation. In 1982, Hazelton (32) improved the first three protocol layer designs. He developed the basic software for the implementation of the Consultive Committee for International Telephone and Telegraph (CCITT) Link Access Protocol (LAP) at the data link layer and the basic structure of the CCITT X.25 protocol for the network layer.

In 1983, Phister (75) improved the data link layer software, developed the initial network layer software for datagram use, and developed the initial start of the transport layer protocol using the

CCITT X.121 and Transmission Control Protocol/Internet Protocol (TCP/IP). The DELNET standards were also established during this effort and are described in (75:Appen C). The requirements and documentation are discussed in detail in Hobart (33), Geist (29), Hazelton (32), and Phister (75:Appen C) and reviewed for completeness in Chapter Two of this thesis. With the application of the standards and a hardware and software modification to the original work (84) to allow the ring communicate in both clockwise and counterclockwise directions, the DELNET architecture evolved as shown in Figure 1-2. There are a maximum of 16 UNIDs on any one dual ring. UNIDs one through 15 service various local area networks and host users. UNID 0 is reserved to connect the dual rings of UNIDs together. The addressing scheme follows the standards established in (75) and is discussed in greater detail in (75:Chap 2).

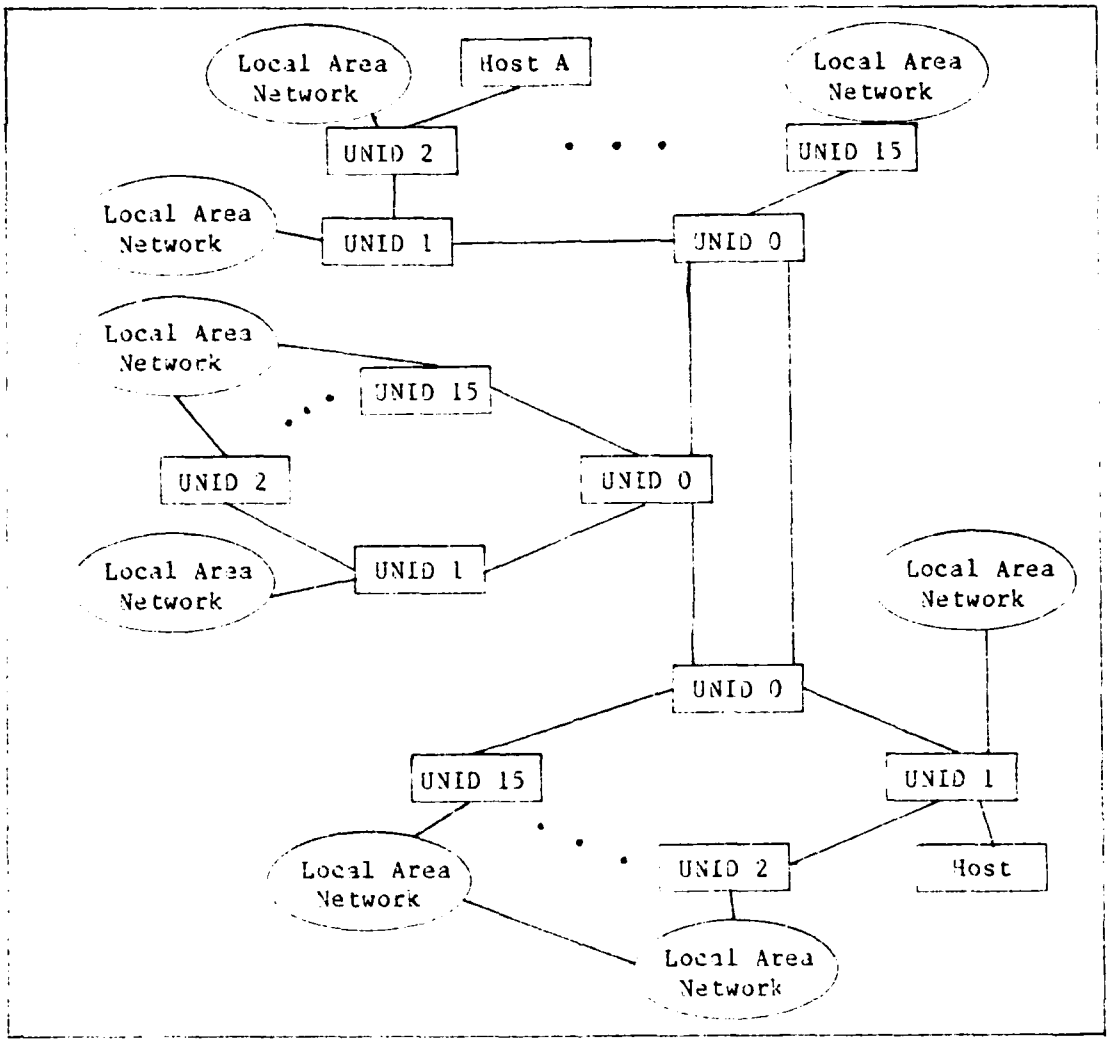


Figure 1-2. DELNET Architecture (75)

Concurrent development of an improved UNID, called the UNID II, based on the Intel 8086 family of processors, began in 1981 when Gravin (30) developed the initial design. He used the current hardware architecture of the UNID I as his starting point. His main goal was to design a UNID II which would be compatible and operate with the UNID I. In 1982, Palmer (73) used Gravin's design and began the hardware and software implementation. While Palmer was able to build both the local host hardware and the network hardware and interface it with an off-the-

shelf 8086 based processor board, the implementation was not completed. In 1983, Matheson (64) followed with the hardware implementation and began the translation of the system software developed by Phister into a language compatible with the Intel development systems. Matheson discovered during his preliminary hardware implementation that the network hardware designed by (73) would not work properly as originally designed (64). He then began an intensive hardware redesign effort to produce a working hardware design. He obtained a feasible hardware design but at the cost of limited software translation from the UNID I system and test of the UNID II system. It is at this point where the current effort begins.

Current Status

A hardware design for the UNID II exists (64). Another alternative using off-the-shelf single board computers was investigated with the result that the UNID II development could proceed more reliably using single board computers. The hardware architectural and implementation changes using Intel SBC 544 and SBC 88/45 single board computers are discussed in greater detail in Chapter Three.

Problem Statement

Integrate the SBC 544 and 88/45 single board computers and implement the existing software designs to produce a functional UNID II. Specifically:

1. Implement the integration of the Intel SBC 88/45 and SBC 544 boards.
2. Convert the network (data link layer) PL/Z software to PL/M and validate its functionality.

3. Validate the functionality of the already translated local (network layer) PL/M software.

4. Integrate the local (network layer) and network (data link layer) software and test the UNID II as a functional entity.

5. Integrate into and test the UNID II in the DELNET.

Scope

The SBC 88/45 and SBC 544 single board computers will be checked for functionality. Development and implementation of the necessary software to validate the functionality of the SBC 544 and the SBC 88/45 boards as a single functional entity will follow. Conversion of the remaining UNID I PL/Z software to PL/M and its functional validation is the next step, followed by a functional simulation of the UNID II software on the Intel ISIS software development system. Integration of the local and network software and its functional validation on the UNID II is next with the integration and test of the UNID II in the DELNET is the last step.

Assumptions

1. It is assumed that the local and network boards work properly.

2. It is assumed that the local software design as previously developed and translated functions properly.

3. It is assumed that the network software design as developed functions properly.

Summary of Current Knowledge

The most up to date summary of the UNID and the DELNET is described in detail in Phister (75). The background in this chapter gives a chronological history of the development of the UNID. The current

status of the UNID II is described above. Detailed information may be found in the bibliography.

Standards

The standards used in this thesis effort will follow those used for the DELNET and the UNID I as developed by Phister, et al, in their work. These standards include:

1. ISO Open Systems Reference Model DP-7490.
2. CCITT X.1, X.2, and X.95 for Class of Service.
3. CCITT X.121 for routing control.
4. Transmission Control Protocol/Internet Protocol (TCP/IP).
5. CCITT X.25 for network control.
6. ISO 3309-1976(E) for data link control.
7. High level Data Link Control (HDLC) protocol.
8. CCITT LAPB.
9. RS-232C, RS-422 and RS-449 at the physical layer.

The standards are international and national in their scope and application and will be followed as previously implemented to maintain compatibility with the UNID I and other data processing, teleprocessing, and telecommunication equipments. The standards will reviewed in further detail in Chapter Two.

Approach

The first step will be to simulate the functionality of the local (network layer) and network (data link layer) software modules. This will be accomplished on the Intel System III in the Computer Communications and Networks Laboratory. The input/output (I/O) requirements of the software can be satisfied by the use of the ISIS

operating system functions. The software can then be validated and corrected as necessary without the use of the target hardware. This approach allows a `simpler' and time efficient software development and validation phase.

The second step is to incorporate the target system initialization software into the local and network software developed in the first step. A final software module is built which can then be programmed into EPROMs for the SBC 544 and SBC 88/45 boards. A CP/M system will be used to simulate a local host for the software validation hosted on the single board computers.

The third step is to integrate the UNID II into the DELNET and begin the software validation of the UNID II software in its intended environment.

Equipment

The Intel System III is required for PL/M software conversion, development, and system software simulation. The Intel System III/210 is required for word processing. Wirewrap tools, wirewrap wire, soldering iron and solder, assorted pliers and other tools are required.

Other Support

Supply and contracting support for the acquisition of integrated circuits, hardware, software and other manufacturer's manuals will be required. A workbench, desk and administrative supplies are required.

Conclusion

The chapter began with the history of the Digital Engineering Laboratory Network (DELNET) and the Universal Network Interface Device

(UNID) I. It then covered the current status of the UNID II, problem statement, scope, assumptions, standards, approach, equipment, and other support required for the further development and implementation of the UNID II to a working system. The next chapter discusses the UNID II and DELNET requirements.

II. UNID II and DELNET Requirements

Introduction

This chapter summarizes the requirements established in previous thesis work (30, 75, 64). A summary of the UNID II requirements is first followed by a summary of the DELNET requirements. This summary is repeated in large part from Matheson's work as no new requirements have been established nor deleted except as noted later in the chapter.

UNID II Requirements Summary

The original UNID design was based on the following general criteria (84):

1. The UNID should function as a store and forward concentrator and have message routing capabilities.
2. The UNID might require specialized I/O ports for unique communication requirements.
3. The UNID should be capable of interfacing to various network operating systems and protocols.
4. The UNID should provide an environment for computer communication network studies.

These concepts are still valid and are the primary design goals for the UNID II. The UNID II is projected to be used in the DELNET. The UNIDs are configured in a dual ring with the host systems forming a star at each node. However, the UNID should be designed to interface with other network configurations which could be implemented at a later date. The UNID hardware should be as flexible as possible so that changes in network protocols can be done in software or firmware rather than by changing or redesigning the hardware.

At the lowest, or hardware, level of protocol the interfaces have been defined to conform with the EIA RS-232C (18) and RS-449 (19)

standards. The local interfaces to computers or terminals will use RS-232C and the network interface between UNIDs are configured for RS-422 and RS-449. Higher levels of protocol should interface so that changes in the upper levels can be accommodated with changes in UNID software rather than hardware. The various levels should have clearly defined interfaces to make updates and changes easier and faster.

Structured analysis and design techniques (SADTs) were used to develop the functional requirements of the original UNID (84). A design using a modular approach was then developed. Three separate hardware modules were identified: (1) a local input/output (I/O) module for interfacing the UNID to the user's computers, terminals, or modems; (2) a network I/O module for interfacing the UNID to other UNIDs over the network; and (3) a dual processor module for matching the local I/O to the network environment (84:154-155). The three module types were selected after analysis of the requirements using SADT methods (84:11-31).

In 1981, a thesis effort was begun to design an improved UNID, the 8086 based UNID II. The original functional requirements (84, 32) were used to produce data flow diagrams (DFDs) (30), and a new functional requirements model was developed for the UNID II. The requirements which were used to develop the requirements model are listed in Table II-I. The result indicated that two distinct groups of requirements were present. One group dealt with the handling of local messages and the other with the handling of network messages. While there were many similarities in function, both groups were considered necessary. The DFDs served as the basis for the design of the UNID II and are of sufficient detail to aid in the implementation of the UNID II. The

original DFDs are reproduced in Appendix A.

Table II-I. UNID II Requirements (64)

I. Interface a wide variety of network components and handle various topologies.

   A. Accommodate dissimilar computing equipment
      1. Accomplish code conversion
      2. Perform data rate speed conversion

   B. Interface peripherals and user terminals to the network

   C. Interface host computers to the network

   D. Provide a network to network interface (a gateway)

II. Perform independently of network components

   A. Handle network data transmission and reception
      1. Accommodate network throughput requirements and flow control
      2. Adapt to different protocols
         a) Handle both synchronous and asynchronous communication
         b) Edit and pack characters into formatted messages
         c) Unpack a message
         d) Perform parallel to serial and serial to parallel data conversion
         e) Handle error control functions such as message acknowledge, no acknowledge, repeat, and timout
      3. Perform error checking and recovery procedures

   B. Relieve host computers from network specific functions
      1. Provide a buffer to smooth message traffic
      2. Poll communications lines if they are multidropped
      3. Handle interrupts
      4. Route messages to desired destinations
      5. Collect performance, traffic, and error statistics

III. Provide a testbed for computer network studies and research

DELNET Functional Requirements

A survey of potential DELNET users was taken in 1981 as part of another thesis effort (32). The responses to the survey were used to formulate a set of functional requirements for the DELNET. A summary of the requirements which were considered to be the most important are:

1. Ability to transfer files across the network.
2. Ability to share peripherals attached to the hosts on the DELNET.
3. Flexibility with respect to the network topology, protocols, and transmission media.
4. Performance monitoring capability.
5. High percentage of availability.
6. User transparancy to network configuration and specific operating systems of hosts.

Other additional features were identified in the survey but were not considered as important for the initial implementation. Some of the identified features which may be considered in the future include:

1. Permit software tool sharing.
2. Perform distributed processing.
3. Use distributed databases.
4. Incorporate fault tolerance.
5. Provide a means to connect to other networks such as the AFITNET and ARPANET.
6. Connect to the local Cyber 750 and DEC VAX 11/780 (Unix).
7. Provide data privacy.
8. Provide security for classified projects.

While providing security for classified projects is a desirable goal, it is not within the scope of the DELNET nor the UNIXs to accomplish this goal. Air Force and other security directives

implemented by the Electronic Security Command (ESC) will not approve a secure network in the AFIT environment due to: lack of physical security; lack of TEMPEST approved equipment; and lack of trusted computer software, and it's associated hardware, validated and approved by the National Security Agency (NSA). Providing data privacy, including user authentication and verification procedures, for the DELNET users, also desirable goals, is within the scope and pervue of the DELNET and the UNIDs. Recommendations in this area are discussed further in Chapter Six.

Utilizing the user generated list of functional requirements, a set of requirements for the DELNET hardware and software was established. A ring topology was initially selected for the DELNET connections with each node providing a star subnet to the local users. This is the same basic configuration recommended in the 1342 EEG Technical Report (1) for base level communications systems except for the star subnet. The report seemed to indicate a single user at each network interface while the DELNET requirement is for multiple hosts, up to four in the current design, on each network node. With the ring topology, development of routing algorithms is easier and system expansion simplified (37:Chap 7), since an elaborate routing scheme is not needed and new nodes can be easily connected to the network.

The system requirements outlined in the original thesis (84) evolved and became the basis for a series of additional investigations. Further refinement of the DFDs was accomplished (33), followed by implementing standards, developing software procedures and writing the necessary code (29, 32, 75, 64). At the same time, the UNID hardware design was modified and refined (4, 11, 74, 83) to correct known

problems, improve reliability and create the dual ring topology of the UNID network (75). Demonstrations of the DELNET were accomplished in 1981, 1983, and 1984 but were limited due to the lack of complete software above the network layer.

Protocols

In establishing the protocol standards for the DELNET, various protocols recommended by both national and international standards organizations were reviewed and those which seemed to 'best' meet the stated requirements were selected. A packet switching protocol using the CCITT X.25 standard was originally chosen (29).

Since the work was and will continue to be accomplished in stages, with flexibility remaining a design goal, a recommendation was made and accepted (32) that the ISO OSI Reference Model (12, 37) be used for the overall DELNET protocol design. The development of specific protocols for the DELNET and implemented on the UNID are (75):

1. CCITT X.25, LAPB (HDLC) in the data link layer.
2. CCITT X.25, datagram service in the network layer.
3. CCITT X.121, internet descriptions in the transport layer.

4a. TCP/IP in the transport layer for datagram service.

b. Federal Information Processing Standards (FIPS) and National Bureau of Standards in the transport layer for virtual circuit service.

Since the UNID II will be nodes in the DELNET, it will support the ISO OSI Reference Model. Previous work (75) established that the UNID would initially support a datagram service, with the three lower layers of the model, called the subnet, implemented in the UNID. The host would implement the upper four layers. This particular separation of

the layers is common for a datagram service network (87:Chap 3). As an aid to understanding both the model and its implementation on the DELNET and the UNID II, a brief description of the model and the above standards follow.

ISO Reference Model

The ISO OSI Reference Model is intended to be a vehicle for the development of specific standard protocols within its seven layers as shown in Figure 2-1. The layering divides a very complex task into smaller tasks with each being relatively independent of the others. Other organizations, such as the National Bureau of Standards, have established and further clarified standards which operate within the framework of the OSI model (21, 22, 23, 24, 25, 26, 27).

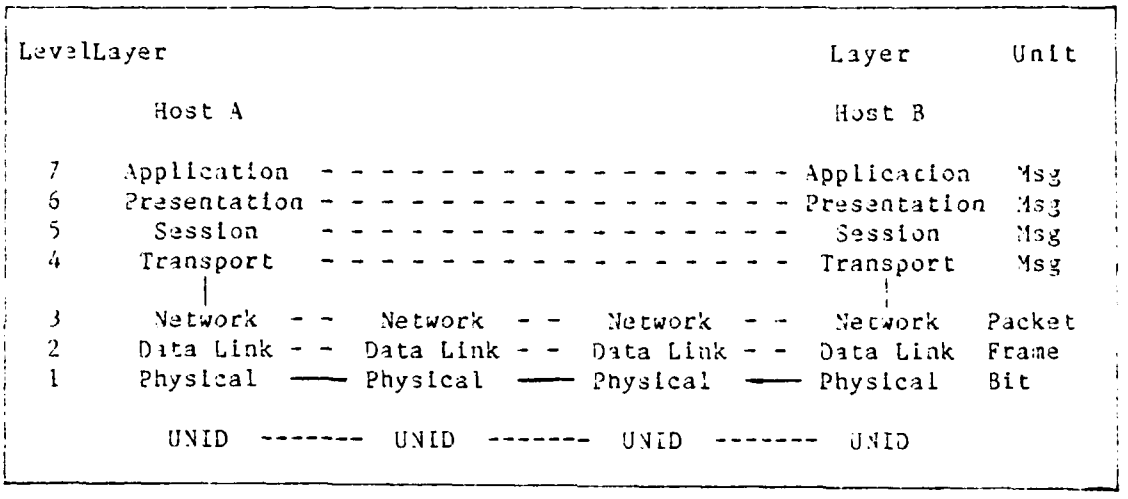


Figure 2-1. ISO OSI Reference Model Applied to DELNET and UNID (75)

At any given layer (except the physical layer), a program in one layer communicates with a corresponding program in the same layer in another host or node in what is called a `peer process'. While the two hosts logically communicate directly with each other (the dotted lines

in Figure 2-1), all communications in fact must pass through the lowest layer since the only physical connection is at that layer (the solid line in Figure 2-1). Each layer, while logically communicating with its peer process in another host, provides various services to the next higher layer above it. For example, the physical layer provides the physical communications link services for the data link layer.

The content of the applications layer is determined by the user requirements. This layer may represent data base exchanges, user to user interactive traffic, or message traffic as examples.

The presentation layer handles the data format transformations which can include data compression, file translation, end to end encryption, and virtual terminal protocols.

The session layer normally performs addressing and connection management of the network for the host, but in fact, some of these functions are often subsumed in the transport or presentation layers in an actual implementation (37:Chap 3, Chap 9).

Once the data is formatted and addressed, it must be transmitted. The transport layer provides the host messages to the communications facilities for eventual transmission. This layer should provide error free, end to end communications between hosts. Some examples of services at this level include error checking and recovery, flow control for the host, sequencing of the messages, and establishment and termination of a host to host connection. The TCP is implemented at this layer in the UNIDs and the DELNET (75: Appen C).

The IP protocol is usually implemented between the transport and network layers to interface networks with different protocols and data entity formats through a common, standard protocol (37: Chap 3). The

most common implementation of the IP is where two different networks are connected through a gateway. A gateway is a set of computer hardware and software programs through which two different networks can communicate. The most common implementation of a gateway is where the functions of the gateway are divided in half and implemented at nodes of the two different networks. The IP protocol is implemented as the common protocol between the gateways and the next higher and lower layers, the transport layer and the network layer, respectively. The IP protocol is effectively sandwiched between the transport and network layers, forming another layer that could be called the internet layer, layer three-and-one-half. The sandwiched layer representation is shown in Figure 2-2 as applied to the ISO OSI in the UNID and the DELNET. Although the IP is used between two different networks, it may also be used between a host and the host's servicing packet switching node. The IP protocol is implemented in the UNID and the DELNET (75: Appen C).

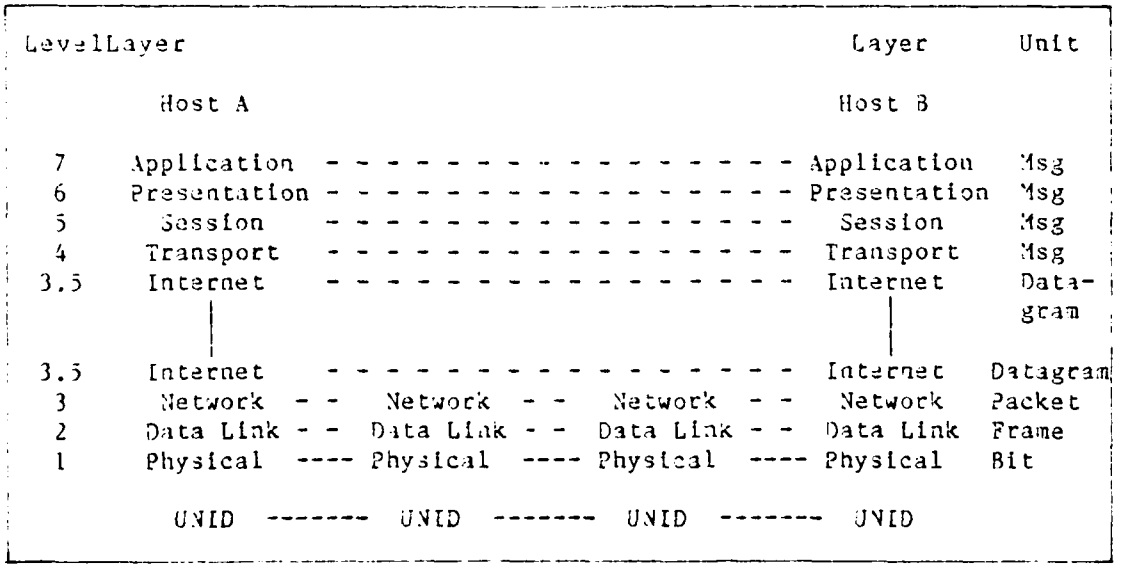


Figure 2-2. ISO OSI Reference Model with the Internet Protocol (IP).

Subnet

The lowest three layers comprise the subnet. These layers route data through the network from one host to another while the higher levels are concerned with the dialogue between the communicating host peer processes. In the DELNET and UNID applications, the UNID contains the subnet protocols. The access protocol selected for the subnet is the X.25 recommendation of the CCITT. This standard was chosen to establish and maintain compatibility with other networks. This protocol standard describes the interface and procedures for packet switched service and is defined in three independent architectural levels which are commonly used for the three subnet layers. Each of the subnet layers is discussed in some detail because they encompass the software and hardware functions of the UNID.

Network Layer

The network layer, referred to as the packet level by the CCITT, is the top level of the subnet and is primarily responsible for routing, sequencing and flow control. It determines the path that a message, or packet, should take through the network from the originating host to the destination host. In most networks the two host computers may be separated by nodes which are not directly connected in the particular connection. There is usually more than one path between the two connected hosts, as well. Each node in the network must determine which way to send the data so that it will reach the intended destination.

In the 1980 revision of the X.25 standard, some significant technical enhancements were made (28). Two of the most important to the DELNET and the UNID are: the addition of provisions for datagram service

and the addition of a fast select facility to the virtual call service.

Datagrams are self-contained packets which contain sufficient address information to be routed to their destinations without the establishment of a virtual circuit through the subnet. Virtual circuit call establishment procedures, with their attendent overhead, are not needed since the datagram is considered a complete message unto itself and independent of all others in the network. The fast select facility provision allows a full 128 bytes of user data to be exchanged during the call set up and clearing procedures for a virtual call. A more detailed description of these services can be found in the literature (28, 75, 37).

The network layer must also deal with congestion. The network can become overloaded if the hosts initiate data into the network faster than it can be processed and delivered. Some method of controlling the amount of data in the network has to be used. One of the more common methods is the exchange of flow control messages with the network layers of other nodes. These messages can include information such as acknowledgement of receipt of data, the number of free message buffers, or the fact that the node is unable to receive data at the present time (37). These features are not as yet implemented in the UNID.

Messages from the transport layer may have a priority, and if so, it is the task of the network layer to insure that higher priority messages are handled first. It must also deliver high priority messages to the transport layer first before lower priority messages are delivered. This aspect of the network layer is particularly important in military and some commercial communications systems where a hierarchy of priorities exists.

Data Link Layer

The data link layer combines groups of bits into logical units called frames. It is the function of the data link layer to create, recognize, and control the flow of the logical bits transferred to and from the network and physical layers. The bit oriented link access control procedure specified in the X.25 standard is the Link Access Procedure B (LAPB), which is equivalent to the ISO High level Data Link Control, HDLC, standard (28). The frame format specified by the standard and implemented in the DELNET and the UNID II is shown in Figure 2-3. The figure represents a composite of the HDLC protocol, X.25 protocol with the datagram service option, and the TCP/IP used in the software implementation of the UNIDs I and II.

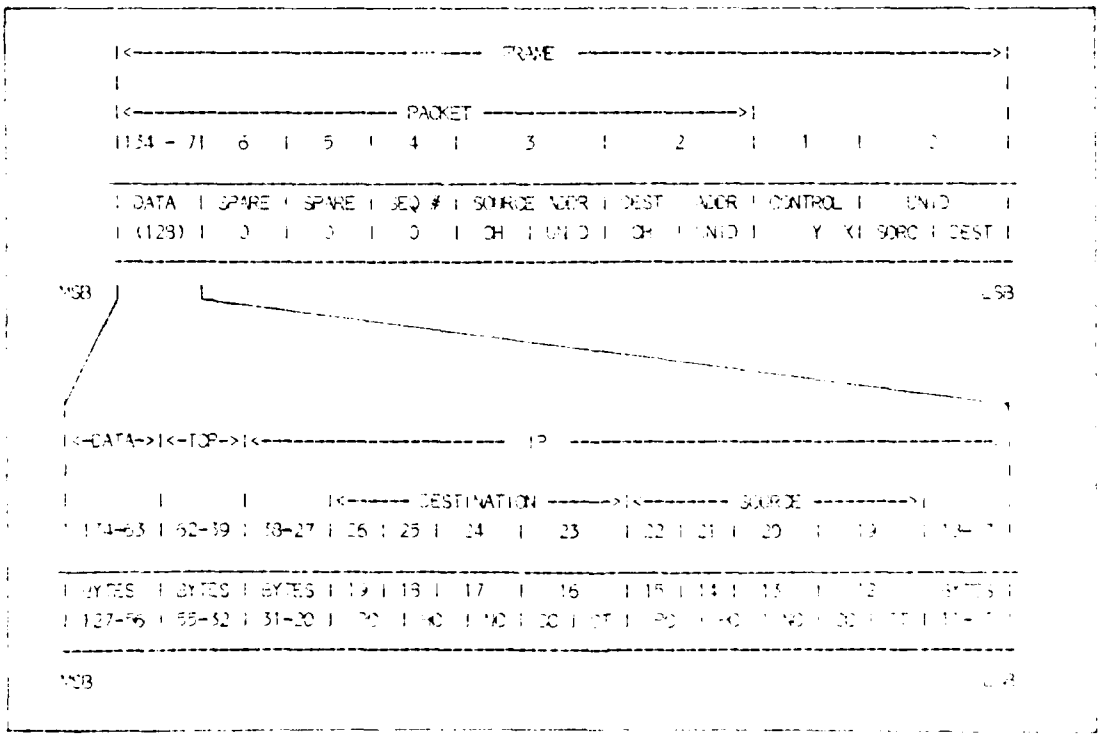


Figure 2-3. Frame and Other Header Information

The TCP/IP, developed for standardized use in the Department of

Defense computer networks and a required protocol for DoD computer networks, is partially implemented in the transport layer of the UNIDs I and II. The source and destination addresses are shown for each of the frame, packet, and datagram where each structure is broken into smaller segments to show its respective contents. The control (CT), country (CC), network (NC), host (HC), and port (PC) codes are shown for the implementation of the CCITT X.121 standard in the IP header used with the UNIDs. A more detailed description of the header structure and contents is in Appendix D and (75: Appen C).

Since the data link layer can receive bad data from the physical layer, an error detection capability is included. A set of 16 bits based on cyclic redundancy check (CRC) calculations, often called a checksum, is appended to the address, control and information bits as they are sent. This checksum is compared with a locally generated checksum at the receiving node. When the checksums match, then it is assumed that the received frame is correct, otherwise the receiver knows that an error has occurred during transmission and the transmitter is notified to retransmit that particular frame. The details of CRC calculation are explained in the X.25 standard (23), while the mathematics involved are explained in the literature (87, 62). Figure 2-3 does not show the flag or checksum bits, even though they are present, as these are automatically calculated, added, and deleted by the digital transmitter and receiver hardware at the network physical level. The flag bits are also appended to the data link layer frame. These flag bits are used for synchronizing the hardware to the beginning and ending of the data. Both the flag and the CRC bits are usually appended and deleted automatically by the physical level hardware. This

hardware method is implemented in the UNID II hardware.

Physical Layer

The physical layer is the lowest level in the network. It provides the physical, electrical, mechanical, functional, and procedural services to define the physical connection between network nodes. The physical interface standard referenced by the CCITT for the host computer to network node is the X.21 digital interface standard. The standards between network nodes are not defined by CCITT for the X.25 standard, therefore, the RS-422 and RS-449 standards are used in the DELNET between the UNIDs to maintain standardization. The host computer or user device is defined as the data terminal equipment (DTE) and the network node (the UNID) is considered data communications equipment (DCE). While the X.21 standard is not widely implemented nationally or internationally due to the lack of true digital communication systems, the X.21(bis) is often used in its place. The X.21(bis) standard is used to interface digital equipment with analog communication systems. While the United States generally does not use the X.21(bis) standard, the RS-232C standard is most often used in its place as the RS-232C standard is functionally equivalent to the X.21(bis) standard (37). The RS-232C standard is the standard implemented between the host and the UNIDs in the DELNET. The original work of Sluzevich (84) included a 20 milliampere current loop as a highly likely physical interface between the UNID and certain host equipments. This requirement, while valid in 1978 and earlier when a considerable amount of equipment using this interface existed, is no longer valid and will not be implemented on the UNID II. Most of the 20 milliampere current loop equipment has been

replaced with newer, more advanced equipment using the RS-232C and MIL-STD-188 standards, even in military communication environments. Appendix B shows the actual RS-232C and RS-422 signals used in the DELNET and UNID II.

Conclusion

This chapter summarized the requirements for the UNID II and the DELNET. The ISO OSI Reference Model and X.25 protocol standard were introduced and their correlation to the UNID II functions was briefly explained. The requirements in Table II-I and data flow diagrams of the functional requirements model in Appendix A form the basis for the design and implementation of the UNID II. Chapter Three discusses the UNID II hardware design and implementation.

III. UNID II Hardware Design

Introduction

This chapter deals with the hardware design of the UNID II. The previous designs (30, 73, 64) are first reviewed to show the design evolution. The current design is then discussed in light of the past designs. The strengths and weaknesses of each design are reviewed.

Initial Designs

The initial UNID II design began in 1981 (30). The investigation centered around several possible configurations for the implementation of a UNID II using the Intel 8086 family of processors as a successor to the UNID I, which was based on the Zilog Z80 processor. The 8086 family of processors was investigated for use with the UNID II because of their increased processing power and ability to generate relocatable code. The final preliminary design is shown in Figure 3-1 (30:Fig 3-4). This configuration has two hardware subsystems, the network and local subsystems, to allow each subsystem to be relatively independent of the system multibus. The network subsystem consists of the 8086 and 8089 processors with the associated input/output (I/O) hardware, private bus interface circuitry and memory and multibus interface circuitry. The local subsystem consists of the four channel serial I/O and a 8086 processor. In addition to the general system requirements (see Chapter Two of this thesis), the design was guided "To allow the network subsystem to be relatively independent of the system bus, (therefore) the network subsystem contains an 8086 CPU and slave 8089 (30:63)." This design allowed network I/O to be handled by one subsystem while the local I/O would be handled by the other subsystem. The design also

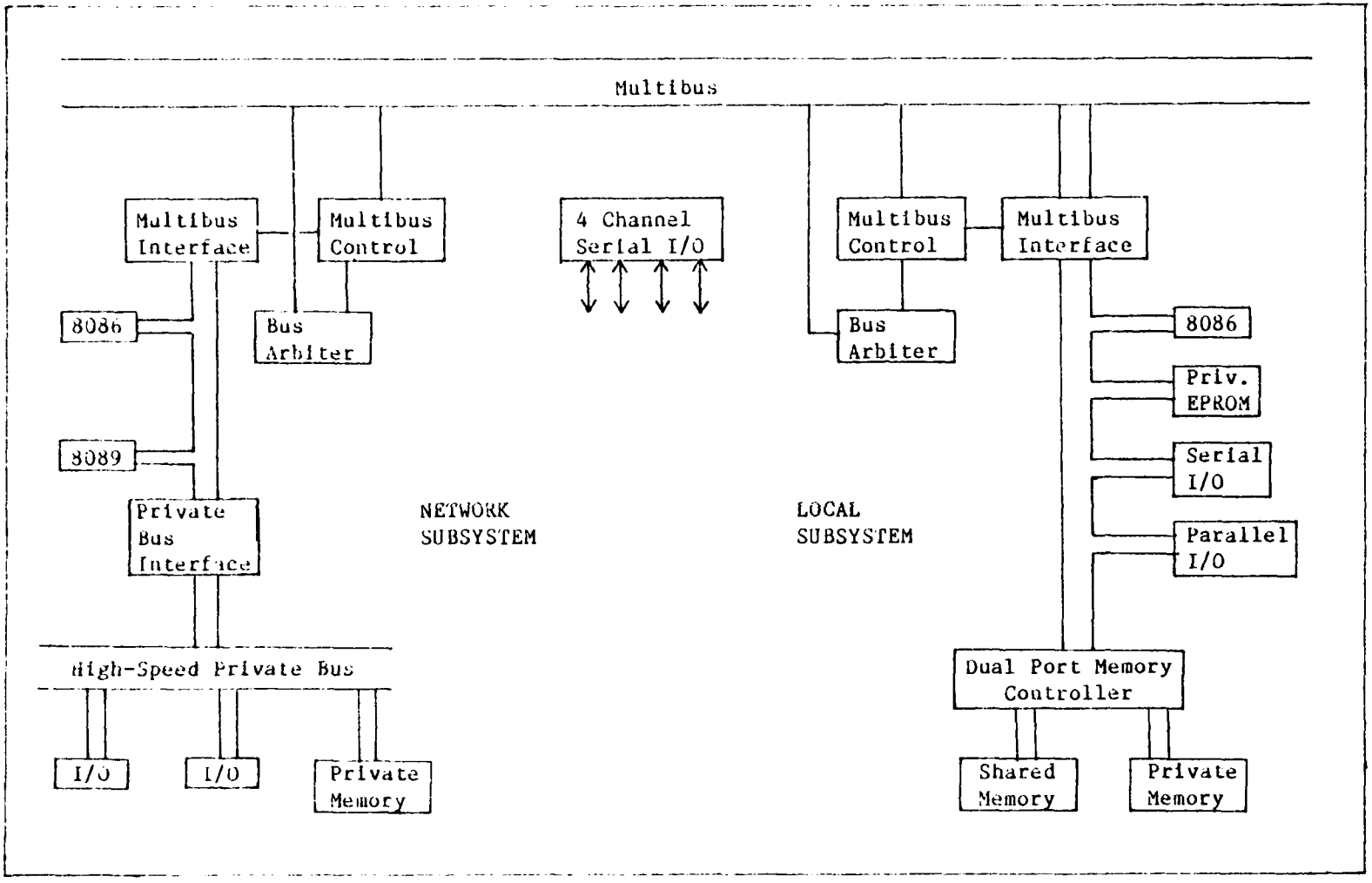


Figure 3-1. UNIO II Block Diagram (30:43)

allowed a network and local software to be physically as well as logically separated and hence, added another degree of modularity in the software design. The largest contribution to the UNID II effort, however, was actually twofold. Not only was a completed preliminary design for the 8086 based UNID II provided, but a functional analysis which led to the data flow diagrams in Appendix A (30:26-34) was completed as well. Processor to processor data transfers were specified to be from data buffers in shared memory. These buffers were separate from the receive and transmit data buffers and functioned as a FIFO queue. Pointers and semaphores were included in block headers to synchronize the communication between the 8086 and 8089 processors. The software design is discussed in greater detail in Chapter Four. More detailed analysis may be found in reference (30).

Original Implementation

The actual hardware, as implemented (30), became three circuit cards in an Intel multibus card rack. The implementation followed from the preliminary design in Figure 3-1 'exactly'. The local processor was an Intel 86/12A single board computer. The four channel serial I/O card was interfaced to the SBC 86/12A through the parallel port of the SBC. The 8086/8089 card was a locally constructed wirewrap card which fit in the multibus card cage. The actual circuit design was based on an Intel applications note (53). Data transfer from the local host was through the four channel serial card connected to the SBC 86/12A through the parallel port. The data was then manipulated appropriately and put in system shared memory and then the network card was interrupted to service the data in the shared memory. The 8086/8089 board would then

manipulate the data and send the data to the 8039 and I/O hardware. Note that all the data transfers between the hosts and the SBC 86/12A were through eight bit universal synchronous/asynchronous receiver/transmitters (USART) and the eight bit parallel port of the SBC 86/12A, even though the data was further manipulated by a 16 bit processor. After the SBC 86/12A manipulated the incoming host data, it would send the data to the 8086/8089 card through reserved blocks of memory in the shared memory space. The multiple protocol communications controllers (MPCC), while allegedly capable of handling 16 bit data transfers, can in fact only handle eight bit data transfers (20:5-267). The MPCCs on the market at that time were capable of interfacing with a 16 bit processor, however the actual data transfer was eight bits of data and eight bits of status or control information. The end result is that the data transfers to and from the MPCCs and the processor are still eight bit data transfers. The MPCCs available on the market as of mid 1984 are still, with the possible exception of very specialized integrated circuits, designed with eight bit data paths. The implementation did produce a minimally functioning system, however the progress was hampered by wiring errors which were discovered during the network board checkout phase (73:Chap 4). The only software implemented at this time was for use with checkout and test of the circuit boards. More detail may be found in reference (73).

Revised Implementation

Further efforts (64) continued the UNID II implementation. During the initial review of the design, it was discovered that the original design would not work properly. The main problems were incompatible

address mapping in the 8086/8089 board (64:3-11) and the fact that the 8089 can handle only two external I/O devices. While there are only two MPCCs on the board, each has a transmit and receive section which must be serviced by the 8089. There were essentially four devices to be serviced by hardware that could only service two devices, therefore a second 8089 was chosen in lieu of the 8086 processor. The redesign efforts resulted in a complete hardware design for the network board. The four channel local host board was still interfaced through the SBC 86/12A parallel port. Figure 3-2 shows revised UNID II design (64:Fig 7). A monitor program was put into EPROM for the SBC 86/12A board to eventually minimize use of the in circuit emulator, the ICE 86 (41), and the local side software was translated from PL/Z to PL/M (64). The work concluded with a functional validation of the local subsystem hardware and software. The converted UNID I code was only cursorily validated. More detail may be found in reference (64).

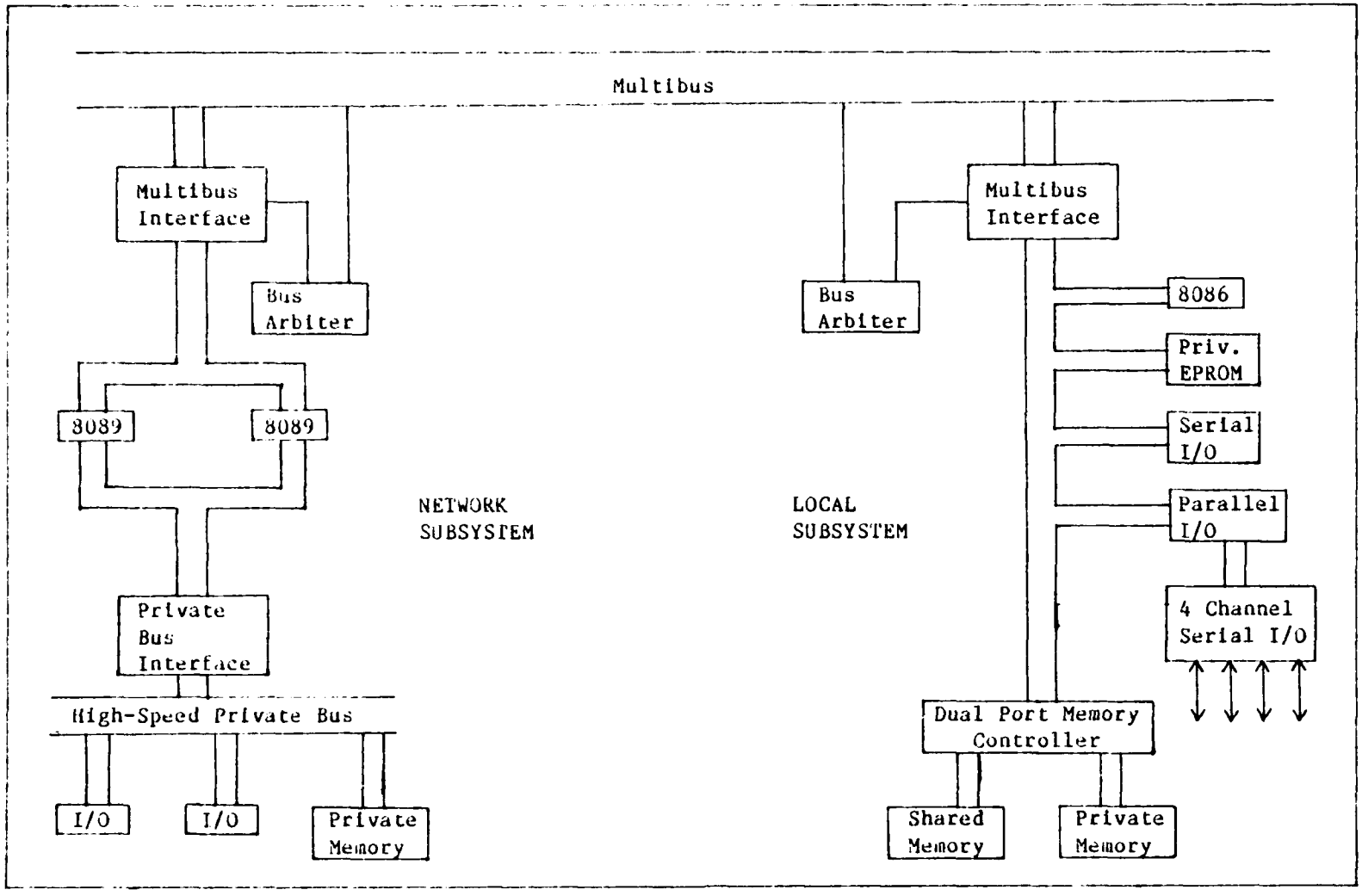


Figure 3-2. UNID II Block Diagram (Revised) (64:1-9)

Current Implementation

The current implementation maintains the same basic architecture as the original and revised designs (30,64). The main difference is in the hardware implementation. The current design utilizes off the shelf single board computers (SBCs) manufactured by Intel. These two SBCs, the iSBC 88/45 and the iSBC 544, are designed specifically for a data communications environment (46, 47). The choice of Intel as a source of the SBCs was due to the SBC availability. Other manufacturers, such as Advanced Micro Devices (AMD), could have comparable hardware although a search of other manufacturer products was not conducted. The choice of the Intel SBCs should not be taken as an endorsement, either expressed or implied, that Intel products are the `best' choice. The selection should be made in light of the system requirements.

The current architecture is shown in Figure 3-3. The heart of the design consists of an 8088 processor servicing two high speed MPCCs through a direct memory access (DMA) arrangement for the network interface and an 8085 processor servicing four interrupt driven USARTS for the local host interface. Both boards have (up to 16k bytes) shared memory and private memory segments as well as Electrically Programmable Read Only Memory (EPROM) (8k to 64k) which was only partially designed in the previous designs. The use of the DMA on the high speed network board fullfills the `DMA-like' operation (39) of the 8089s in the revised design. The interrupt capabilities used on the 8085 based local board simplifies the software design and implementation of the USART supporting software. Each SBC has spare counter/timers which may be used for real time counters, timers, or clocks to support the timeouts required of the network and data link layer protocols. More detailed

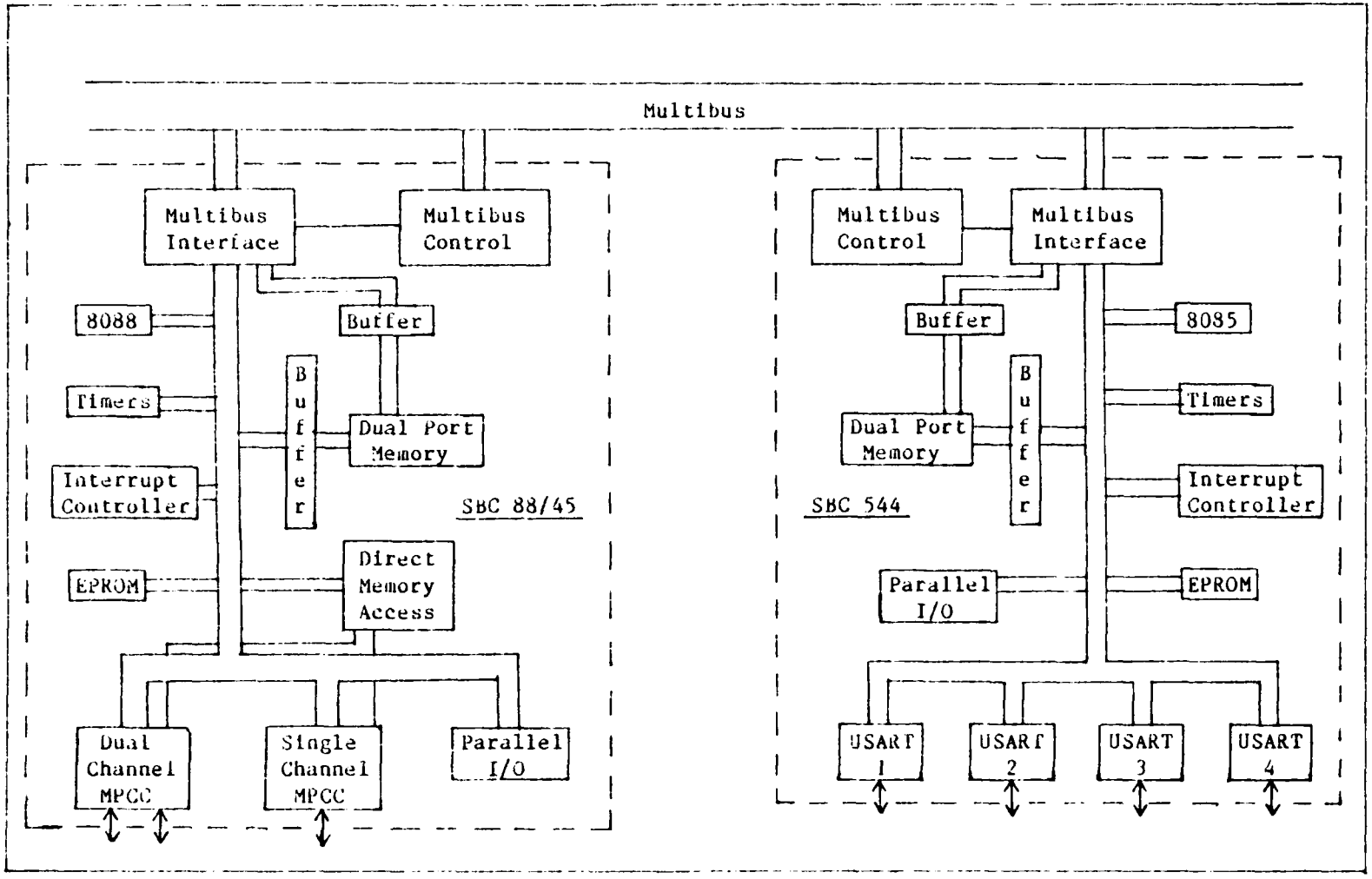


Figure 3-3. Current UNID II Block Diagram

explanation of the SBC may be found in references (46, 47).

The use of these SBCs eliminates the hardware design, implementation, and hardware debug phases of an in-house design effort. The hardware reliability of a known hardware/software manufacturer, the availability of the software development tools (a full screen editor, the PL/M compiler, 8085 and 8086 assemblers), the availability of PL/M80 and PL/M86, both high order languages, and the ease of hardware expansion to support software expansion lead to this implementation decision.

Conclusion

This chapter explained the hardware design of the UNID II. The previous designs (30, 73, 64) were first reviewed to show the evolution of the design. The current design was then discussed in light of the past designs. The strengths and weaknesses of each design were reviewed. Chapter Four discusses the UNID II software design and implementation.

IV. UNID II Software Design and Implementation

Introduction

This chapter discusses the detailed design and implementation of the UNID II software. The discussion is divided into six sections based upon the top-down design approach: Development Language Selection; UNID II Data Structures; UNID II Network Layer Software Design; UNID II Data Link Layer Software Design; UNID II Software Development Tools; and UNID II System Memory Map.

Development Language Selection

The development language used for the the UNID II is a high order language called PL/M, a subset of PL/1. The choice of a high order language versus assembly language follows current software engineering trends: modularity, ease of design and maintenance, ease of understanding, self documenting ability, well defined module interfaces, code and data hiding, and simple parameter passing. A small assembly language module was used, however, in a supporting test program due to the fact that the particular module requirements were satisified by a known working and reliable module. Chapters Five and Six address testing in further detail.

The choice of PL/M is simply pragmatic: the compiler, other software tools, and the development system on which to run them were available in the Air Force Institute of Technology computer networks development laboratory. While both Pascal and C compilers were available for the 8085 processor, neither were available for the 8086/8088 processors at the start of this investigation. And while both Pascal and C could be used in future enhancements of the UNID software

development, this author recommends the use of C instead of Pascal because of C's ability to manipulate data at the bit and byte level and the availability of C on many UNID host systems. Pascal cannot adequately manipulate data at the bit and byte level. Manipulation of data at the bit and byte level is required for the correct interpretation and manipulation of information in the data entity headers. PL/M is more than adequate for use on the UNID hardware, i.e., the SBC 544 and SBC 88/45, however further implementations of the UNID may not use this hardware and hence the software developed with PL/M is not readily transportable to another hardware system. The reader is cautioned, however, that implementations of the UNID software in any language will require that the developer write the low level software drivers for the USARTs/MPCCs. The low level drivers provided with Pascal and C compilers typically interface to a host development system operating system services which do not exist in the UNID software. Low level I/O software drivers may need to be written as well to support the hardware serial link to the UNID.

UNID II Data Structures

As previously discussed in Chapter two, the main purpose of the UNID II software is to move information from one point to another point in a timely and orderly fashion. The means to accomplish that end, while strongly dependent on the system hardware, is also highly dependent on the system software. The DFDs in Appendix A, while outlining the steps to move the data and implying a data structure, do not specify or explain in sufficient detail how the data is to be organized or moved. Since the implementation of the PL/M language lends

itself easily and efficiently to indexed arrays (56, 57) and previous software implementation efforts (29, 30, 64, 75) used indexed arrays, indexed arrays were chosen as the data structure for the input and output buffers from and to the input and output serial ports. The actual indexed arrays are implemented as circular first-in, first-out (FIFO) queues. A FIFO linked list structure could also be used, however dynamic memory allocation for the linked list in the PL/M language is not available and the implementation details of how to implement linked lists is not clear from the available manufacturer's data, therefore, considerable additional effort would be spent developing this type of supporting software. Furthermore, this author's experience with linked lists in the PL/M language is limited, further contributing to a longer development time developing such supporting structures. Where it was more efficient in terms of minimizing memory usage, both for data tables and code size, and increasing processing speed, the data is left in an array and a pointer to the current data is passed to the using modules (11).

Semaphores are used between the local and network SBCs to protect critical regions of executable code and data (13). Appendix E explains in more detail the rational for use of the semaphores and which critical regions of code they protect between the local and network SBCs.

The data structure relationships as implemented in the earlier development efforts (32, 64, 75) are shown in Figure 4-1. The basic structure is an indexed array, or table, as implied by the DFDs in Appendix A. There are tables for each receive (LCOxRX, NTOxRX) and transmit (LCOxTX, NTOxTX) port as well as common tables (LCNTTB, NTLCTB) in shared memory. In addition, the original design (32) included the

tables (LCLCTB, NTNTTB) between the receive and transmit tables in both the local and network modules. The figure also shows the type of the data entity (datagram, packet, frame) in each table along with the size of that data entity (128, 133, 135 bytes respectively) below the tables. The original design, as well as the current implementation, allows ten data entities per table (32, 64, 75). The number of data entities per table is not based on any analytical techniques or studies but rather picked as a reasonable number of entries per table for the initial implementations. Exact table sizing is discussed in the recommendations section of Chapter Six. The last pair of lines indicate where the applicable OSI layer fits into the data structures. The solid lines connecting the tables indicate the data flow between the data structures.

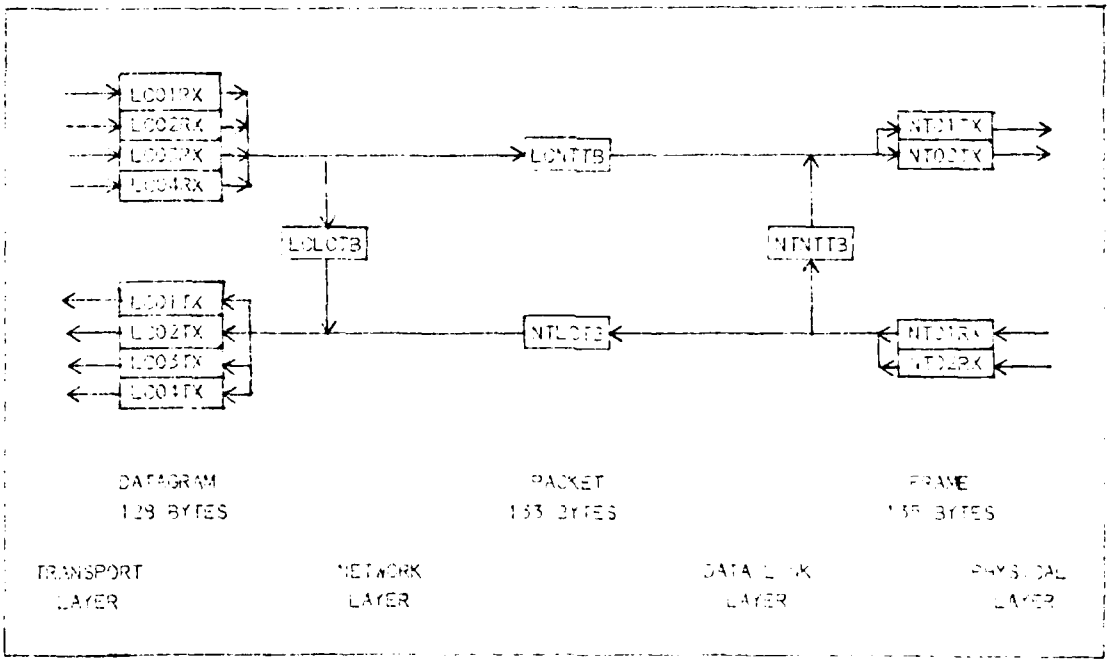


Figure 4-1. Original UNID Data Structures and Flow.

The modified data structures as implemented in this development

effort are shown in Figure 4-2. The indexed arrays on the transmit and receive buffers of the serial ports have been retained. The receive buffers are in shared memory available to both processors while the transmit buffers are private to each processor. The local-to-local, network-to-network, local-to-network, and network-to-local tables in the original implementation are not required and have been eliminated. They increase both the size of the required code by virtue of their size and increase the processing time to move the data.

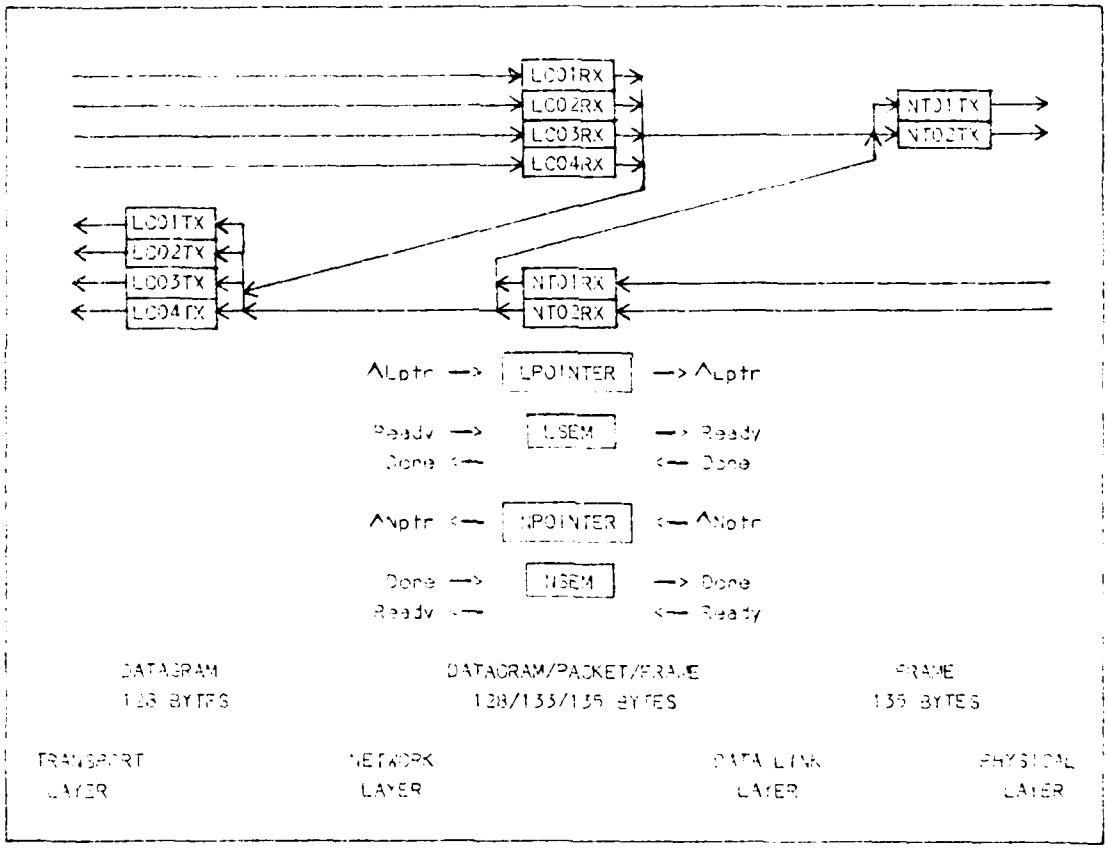


Figure 4-2. UNID II Data Structures and Flow

Pointers (LPOINTER, NPOINTER) to the current data element to be moved (datagram, packet and frame) are passed to the subordinate software modules in order to minimize both processing speed and code

size. Semaphores (LSEM, NSEM) are used to indicate when a data element is ready to be moved. Appendix E explains their structure and use in more detail. Data element movement occurs only when an element must be moved from the receive buffer to the transmit buffer. As in Figure 4-1, Figure 4-2 also shows the type of the data entity in each table along with the size of that data entity below the tables. The last lines indicate where the applicable OSI layer fits into the data structures. The solid lines connecting the tables indicate the data flow between the data structures.

UNID II Network Layer Software Design

The basic design of this software module follows the DELNET requirements elaborated upon in Chapter Two. The CCITT X.25 Datagram protocol was partially implemented to gain a minimally working software module (e.g., pass datagrams and packets). The DoD standard TCP/IP (69, 67) protocols and a variation of the CCITT X.121 internet addressing protocol (75:Appen C) were implemented. The variation is a result of studying the X.121 addressing concept as applied to the addressing space available in the IP protocol header. As the reader will recall from Chapter Two, the variant X.121 internet addressing protocol is imbedded in the 32 bit source and destination addresses of the IP protocol. One should also recall that communication between layers is accomplished through the information in the data element header. In addition, the IP, and its supporting software in the UNID and host(s), behave as half gateways interfacing two networks (87:Chap 3).

The basic function of the network layer is to accept datagrams from

an attached host, determine the datagram destination from the IP destination address, and route the datagram either to another host attached to the same UNID or to the data link layer after constructing a packet header for the data link layer. Figure A-1 in Appendix A is the applicable DFD for this level of implementation. The network layer also accepts packets from the data link layer, interprets the destination from the packet header, and routes the resulting datagram to the attached host. A data dictionary is in Appendix G and compiled listings are in Appendix H for the entire set of software programs used in this effort.

Figure 4-3 represent a primitive structure chart showing the high level structure of the network level software.

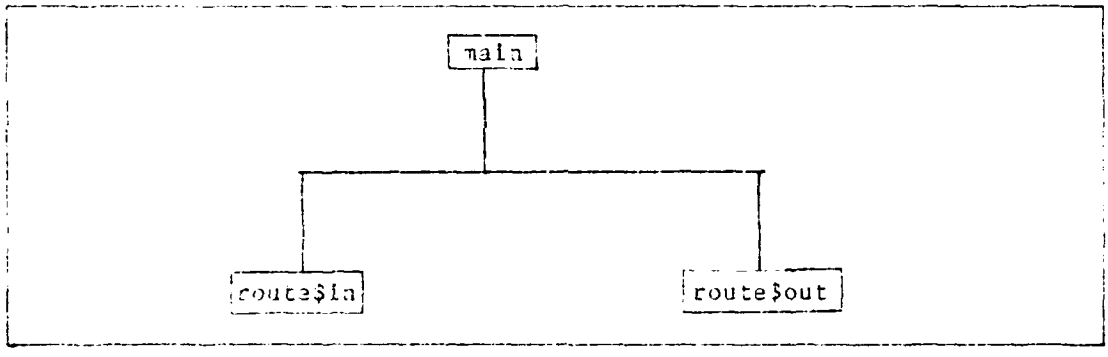


Figure 4-3. Network Layer High Level Structure Chart.

Figure 4-4 shows the structure chart representation for the route$in procedure and Figure 4-5 elaborates the route$in procedure in plain language pseudocode. The `do´-`end´ construct is taken directly from the PL/M language to explicitly delineate a specific block of code. The construct functions exactly as the `begin´-`end´ construct in Pascal. The route$in procedure routes data coming into the UNID to its respective destination. The first part of the code is representative of

the code for each of the four ports on the SBC 544 board with the addition of a differentiating number in the variable names to delineate to which port the code belongs. The second part of the code is representative of the code for each of the two receive channels from the network (data link layer) software. Again, differentiating numbers in the variable names are used to delineate from which network port the packet arrived.

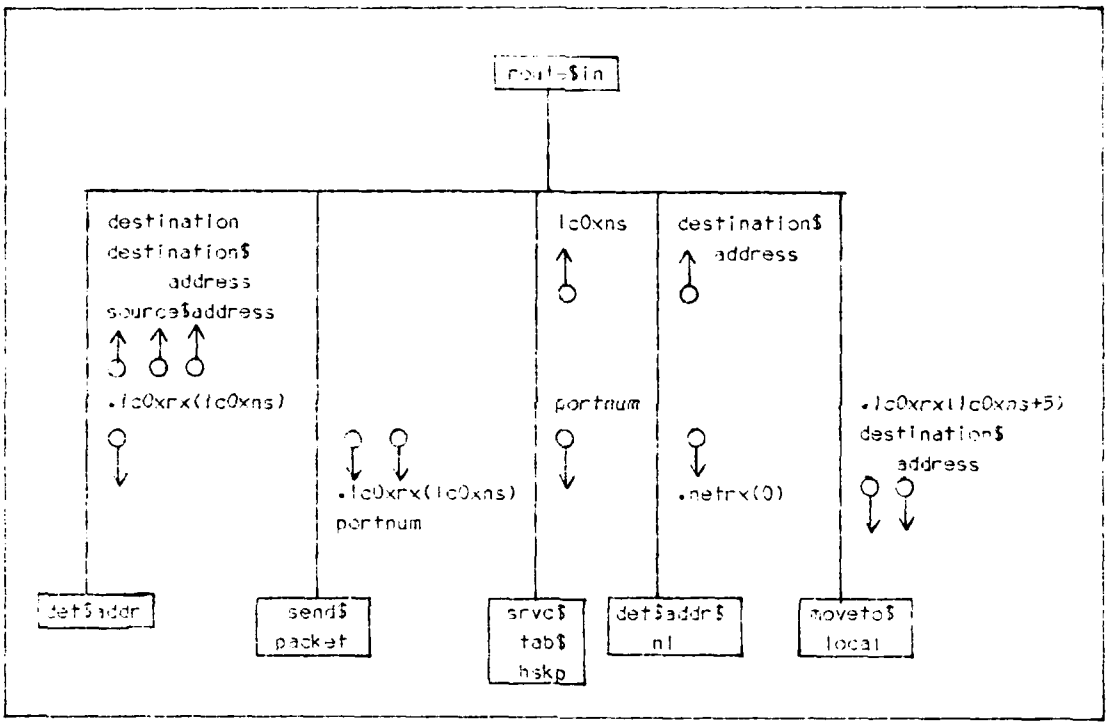


Figure 4-4. Route$In Procedure Structure Chart.

```
if datagram in receive buffer then
     do
     determine destination address
     if destination is back to local host then
          do
          move datagram to local host transmit buffer
          adjust receive buffer pointer
          end
     else
          if destination is to remote host then
               do
               if network done then
                    send packet to network
               end
          else
               do
               increment error count
               adjust receive buffer pointer
               end
     end


if packet from network then
     do
     determine local address
     if legal address then
          move datagram to local host transmit buffer
     set network semaphore done
     end
```

Figure 4-5. Route$In Procedure Pseudocode

The determine destination address (det$addr) procedure interrogates the incoming IP datagram header to determine the destination for that datagram. If the destination is for a UNID other than that servicing the current host, the source address is also determined. This routing information is available through the use of global variables for the send$packet procedure. The send$packet procedure builds the five byte packet header on the datagram information, moves the pointer to a variable in shared system memory, and sets a semaphore to the appropriate state and increments the receive table pointer. Details of the use of semaphores for communication between the SBC 544 and SBC

88/45 boards are explained in Appendix E. If the datagram is destined for a host on the same UNID, only the destination information is determined. The destination information is then used with the move to local table (moveto$local) procedure to move the datagram to the correct transmit table. If an error exists in the destination information, the destination variable is set to a nonexistant destination to indicate an error in the destination. In all three cases above, the receive table pointer is adjusted to indicate the movement of the incoming datagram. If the semaphore from the SBC 88/45 board is not appropriately set, the received datagram is not processed and receive table pointer is not adjusted for local to network data movement. The datagram will be interrogated again when the route$in procedure is next executed.

The second part of the route$in procedure determines if a packet from the network is destined for a local host. If so, then the local host port address is determined. When the host port address is valid, then the datagram is moved from the packet to the host transmit buffer for transmission to the host. The semaphore to the SBC 88/45 board is then set appropriately to indicate the packet has been processed. Note that the semaphore is set to indicate completed processing even though the local address may be in error. Furthermore, the incoming data is not moved to any local transmit buffer if the local address in the packet header is in error. Similar processing occurred with the local host to network movement; datagrams or packets with incorrect destination information are not moved and are effectively `thrown away.' The net effect is that bad data is not allowed to go through the UNID; it is destroyed.

The send$packet procedure fills the packet header with the

destination and source information determined by the determine destination procedure. It then puts the pointer to the current data into a variable in common system memory for the SBC 88/45 and the data link layer software to read. A semaphore is then appropriately set to indicate to the data link layer software executing on the SBC 88/45 board that a packet is ready for transmission into the network. The buffer index is then updated.

The determine local address (det$addr$nl) procedure is used when a packet is received from the data link layer software destined for a local host. The procedure interrogates the packet header to determine for which host the packet is determined. An error in the packet addressing will return a nonexistant address to indicate to the calling procedure that the header addressing is in error and the packet should not be sent to a host and should instead be destroyed.

The move to local table (moveto$local) procedure receives a pointer to the current datagram to move to the host indicated by the determine local address procedure. The procedure moves the data from the common system memory to the local host transmit buffer and adjusts the buffer pointer accordingly.

The companion procedure route$out detects if a datagram is present in the host transmit buffers. Figure 4-6 is a representation of the applicable structure chart.

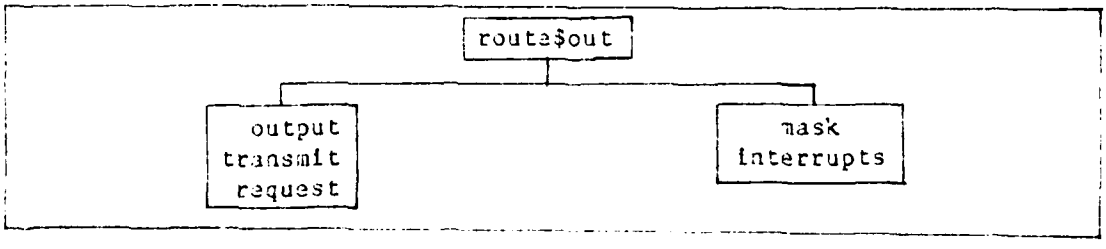


Figure 4-6. Route$Out Procedure Structure Chart

```
disable interrupts
mask receive USART interrupt off
enable interrupts

if datagram available and not sending then
     do
     if TRTA$handshake and (not sending and not receiving) then
          do
          set transmit request true
          set sending true
          send transmit request
          end

     if not TRTA$handshake or (sending and not receiving) then
          do
          set sending true
          disable interrupts
          mask transmit USART interrupt on
          enable interrupts
          end
     end

disable interrupts
mask receive USART interrupt on
enable interrupts
```

Figure 4-7. Route$Out Procedure Pseudocode

Figure 4-7 shows the plain language pseudo code for the route$out procedure. If a datagram is present, then the procedure must check certain boolean flags to determine if a software handshake is in use with the desired host. The term `software handshake' is used here to mean the process by which two communicating programs coordinate and synchronize the sending or receiving of a data entity such as a datagram. Some processors, by virtue of their hardware and software architecture, cannot receive datagrams (or other data entities) on a random basis. These type of systems usually use polled input/output schemes as opposed to interrupt driven I/O schemes. Therefore, this type of system cannot reliably receive or transmit information until it is ready to do so. The means to communicate to another processor that data

transmission or reception is ready to commence is often accomplished through a software handshake. The AFIT LSI-11 Network Operating System (NETOS) is such a system (31, 72) and will be discussed in more detail related to UNID II testing in Chapter Five.

The software handshake mechanism can be described as follows. When a host desires to send a packet to a UNID, it first sends a transmit request (TR) to the UNID. The receiving UNID then, when it recognizes a TR was sent to it, will send a transmit acknowledge (TA) back to the host when it is ready to receive a packet. The sending host, when it recognizes the TA from the receiving UNID, sends the datagram to the receiving UNID. There is no final acknowledge sent by the UNID to the host to acknowledge the reception of the datagram. The TR/TA mecha

The companion procedure route$out detects if a datagram is present in the host transmit buffers. Figure 4-6 is shown in Figure 4-8.

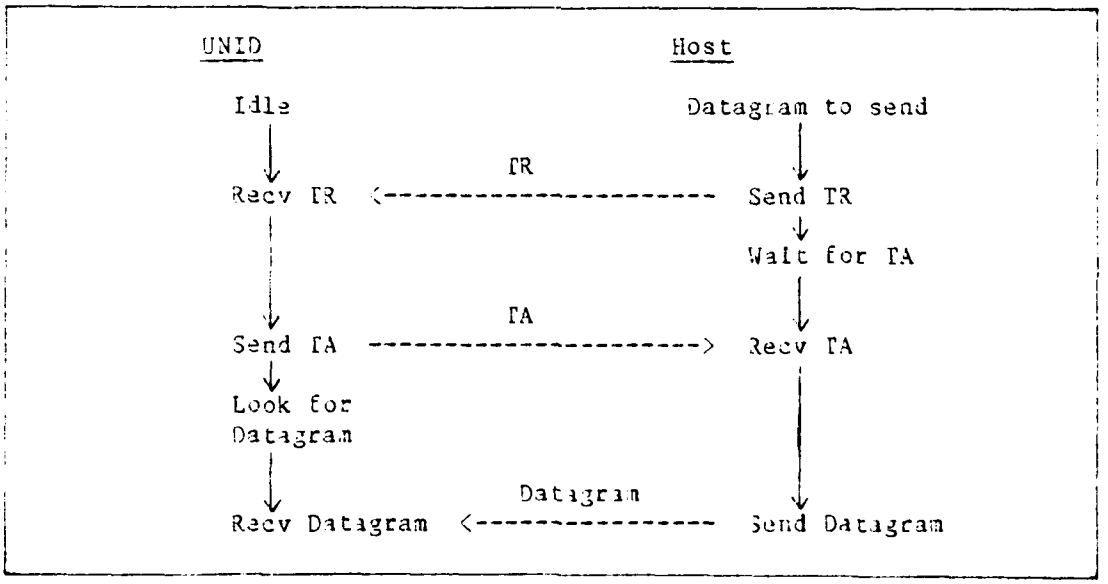


Figure 4-8. UNID/Host Transmit Request/Transmit Acknowledge Handshake.

The handshake mechanism is implemented in the UNID II software with

four boolean flags for each host port. Four flags are required since both the host and the UNID will send and receive datagrams using the TR/TA mechanism, providing a full handshake for each direction. The UNID must know which state it is in so that it can communicate correctly with the host. The four flags are Transmit transmit request (TXTR), Receive transmit acknowledge (RXTA), Receive transmit request (RXTR), and Transmit transmit acknowledge (TXTA). Each flag has the value TRUE or FALSE. The initial state is all four flags FALSE. Of the 16 possible states, the five allowed states are shown in the truth table in Figure 4-9. A '0' represents FALSE and a '1' represents TRUE.

TXTA	RXTA	RXTR	TXTA	
0	0	0	0	Initial state
1	0	0	0	Datagram to send
1	1	0	0	OK to send datagram
1	1	0	0	Send the datagram
0	0	0	0	Reset the flags after sending datagram
0	0	1	0	Datagram receive request
0	0	1	1	OK to receive datagram
0	0	1	1	Receive datagram
0	0	0	0	Reset flags after receiving datagram

Figure 4-9. UNID TR/TA Allowable States.

Note that the mechanism starts in the all zero, or FALSE, state with no datagrams to send or receive and returns to the all zero state at the completion of sending or receiving a datagram. Also, only one

process of sending a datagram or receiving a datagram is allowed at one time.

If the handshake is present, then the boolean flags may only be in certain states before sending the datagram, i.e., the UNID cannot transmit if it is receiving a datagram from the host and it cannot receive a datagram if it is transmitting a datagram to the host. Appendix F explains the use of the software handshake used in the NETOS and in conjunction with UNID II testing.

Note in Figure 4-7 that the interrupts are disabled before masking "off" or "on" of a particular interrupt bit. The interrupts must be disabled before changing the interrupt mask as it is possible that an interrupt could occur which would change the interrupt mask during the time that the interrupt mask is being changed in the route$out procedure. Since the interrupt mask could be changed by an interrupt procedure while another process is attempting to change the mask, the later process is then a critical region and must be protected before any manipulations of the interrupt mask occur (13:78). Further discussion of critical regions may be found in Appendix E and (13).

The companion procedures that communicate with the route$out procedure using the software handshake are the transmit and receive interrupt procedures. Each of these procedures must be capable of sending and receiving datagrams correctly depending on the particular state of the TR/TA boolean states. The procedures must, in addition, be capable of transmitting and receiving datagrams from a host that does not require the TR/TA handshake. The plain language pseudo code for the receive interrupt is shown in Figure 4-10 and the for the transmit interrupt is shown in Figure 4-11. Note that the code, along with the

code of the route$out procedure in Figure 4-7, implements the logic of the four boolean variables in the truth table in Figure 4-9.

```
if ((not trta) or ((rxtr and txta) and ((not txtr) and
        (not rxta)))) then
     do
     put received character in next empty buffer space
     increment received character count
     increment receive buffer index
     if received character count >= datagram size then
          do
          adjust pointer to next datagram area
          if receive buffer index >= max index then
               reset index
          reset receive character count
          reset rxtr false
          reset txta false
          reset send false
          end
     end

if trta then
     do
     if receive character = transmit acknowledge then
          if ((txtr and (not rxta)) and ((not rxtr) and
                        (not txta))) then
               do
               set rxta true
               reset send false
               end
     if receive character = transmit request then
          if (((not rxtr) and (not txta)) and ((not txtr) and
                        (not rxta))) then
               do
               set rxtr true
               set txta true
               set send true
               send ta to host
               end
     end

clear interrupt
```

Figure 4-10. Receive Interrupt Procedure

The implementation of the four boolean variables guarantees that the software will be executed exactly according to the logic in the

truth table. Note also that the routine counts the number of bytes received and increments the index in the buffers. When a full datagram is received, then the pointers and TR/TA variables are adjusted for the next datagram reception. The variable send is used to guarantee that the software does not attempt to send a datagram when transmission of a datagram has already begun. This prevents the UNID from attempting to send the same datagram more than once before a slow host can respond to the initial datagram transmission.

```
if ((not trta) or ((txtr and rxta) and ((not rxtr) and
                    (not txta)))) then
     do
     send next character to host
     increment transmitted character count
     increment transmit buffer index
     if number bytes sent >= datagram size then
          do
          mask transmit interrupt bit off
          reset transmitted character count
          if transmit buffer index >= max index then
               reset
          reset txtr false
          reset rxta false
          reset send false
          end
     end

clear interrupt
```

Figure 4-11. Transmit Interrupt Procedure

UNID II Data Link Layer Software Design

The basic design of this software module follows the DELNET requirements elaborated upon in Chapter Two. The CCITT X.25 LAPB standard (28) was implemented in previous work (29, 32, 75) with one variation which was retained in this work. The variation relates to the number of unacknowledged frames the data layer link software will allow

before retransmitting the next frame to send. The LAPB standard allows three bits, or eight frames, for use with its sliding window protocol (37:148-153). The implementations in the previous work use only one bit and allow only one unacknowledged frame before retransmission in order to gain a minimally working software module that passes frames correctly (29, 32, 75). The use of one bit in this protocol is known as a one bit sliding window protocol (37:151). This work follows previous work and uses only one bit and one unacknowledged frame before retransmission. As in the network layer, communications between layers is accomplished through the header information, in this case with the packet header.

The basic function of the data link layer is to accept frames from the network and route them either to the network layer or to another UNID in the network and accept packets from the network layer and route them to the network. Sequencing and flow control between UNIDs in the network is done at this layer. Figure A-1 in Appendix A is the applicable DFD for this level of implementation. The data dictionary in Appendix G and the compiled listings in Appendix H contain the detailed implementation of this software module.

Figure 4-12 represents a primitive structure chart showing the high level structure of the data link layer software. Packets and frames are routed into and out of the UNID at this layer.

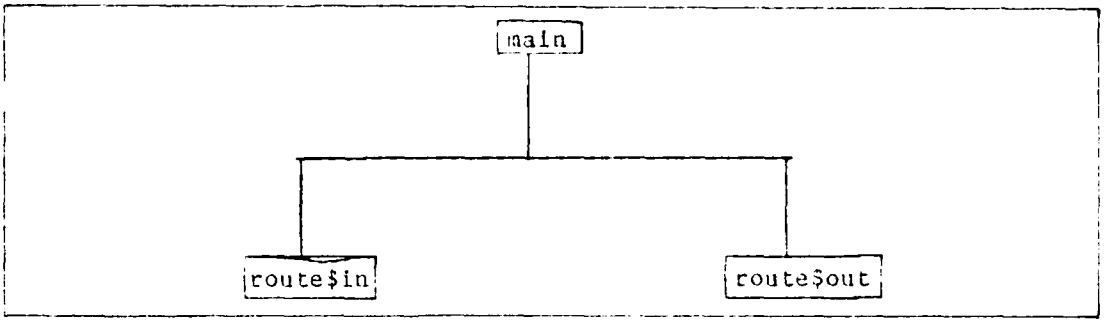


Figure 4-12. Data Link Layer High Level Structure Chart

Figure 4-13 shows the structure chart representation for the route$in procedure. The `do´-`end´ construct is taken directly from the PL/1 language to explicitly delineate a specific block of code. The construct functions exactly as the `begin´-`end´ construct in Pascal.

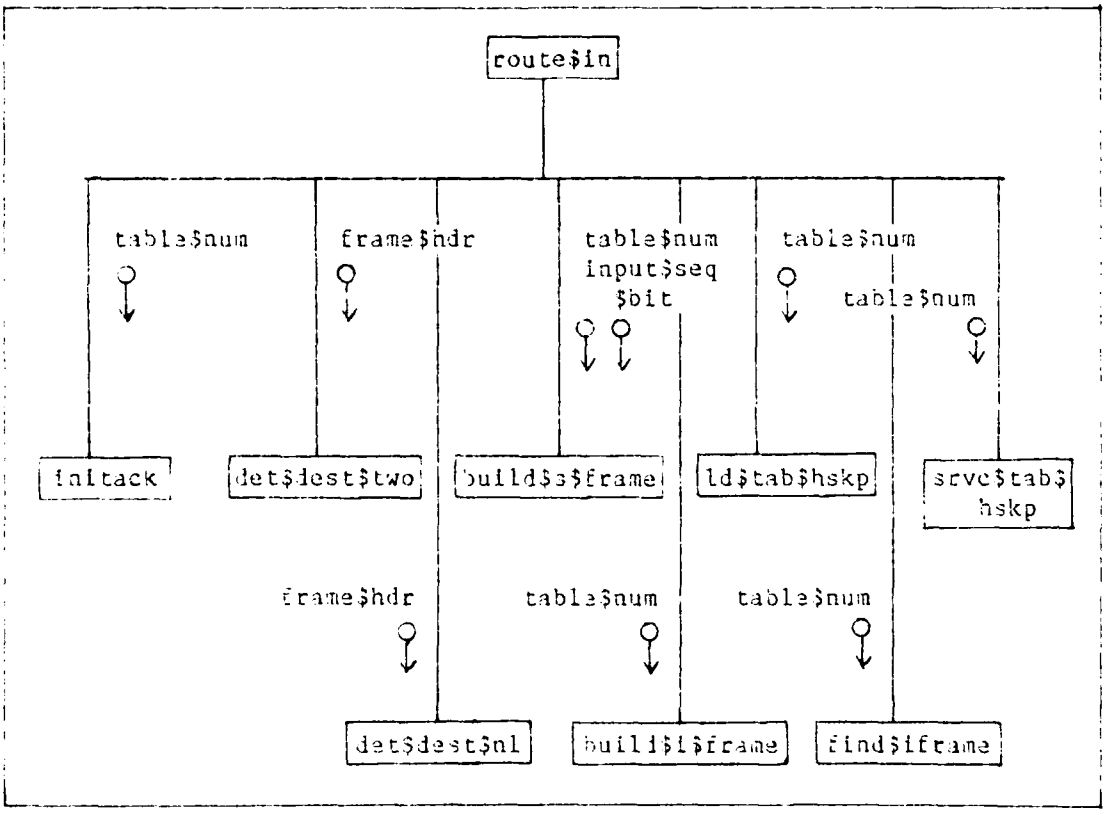


Figure 4-13. Route$In Procedure Structure Chart

```
if a frame has been received from network then
     do
     if the frame is a supervisory frame then
          do
          if the sequence bit is true then
               do
               set this sequence bit true
               if this sequence bit equals transmitted sequence bit
                    then do
                    toggle the transmit sequence bit
                    initialize the acknowledge variables
                    end
               end
          else
               do
               set this sequence bit false
               if this sequence bit equals transmitted sequence bit
                    then do
                    toggle the transmit sequence bit
                    initialize the acknowledge variables
                    end
               end
          end
     else        { frame is an information frame }
          do
          determine destination of information frame
          if destination is network to network then
               do
               if frame sequence bit is true then
                    set input sequence bit true
               else
                    set input sequence bit false
               build a supervisory frame
               move frame to transmit buffer
               service transmit buffer pointer
               end
          if destination is network to local then
               do
               if frame sequence bit is true then
                    set input sequence bit true
               else
                    set input sequence bit false
               build a supervisory frame
               move frame to local buffer
               service local buffer pointer
               end
          end
     end
     service receive buffer pointer
end
```

Figure 4-14. Route$In Procedure Pseudocode (Part 1)

Figures 4-14 and 4-15 elaborate the route$in procedure in plain language pseudocode. Figure 4-14 represents the software for one of the two UNID II data channels communicating with the rest of the network. Only one set of channel software is shown here for clarity.

In Figure 4-14, the software determines if a frame is in the receive buffer. When a frame is received, the frame header is interrogated to determine if the frame is an information frame or a supervisory frame. A received supervisory frame is, in this software implementation, an acknowledgement from the next UNID that an information frame has been received correctly from this UNID. Other supervisory information may be coded in the frame header however these features were not implemented in the current work. A received information frame is data from another UNID destined for another UNID or a host attached to this UNID. The supervisory and information frames must be detected first before the destination of the frame in order to correctly determine the disposition of the received frame. An information frame is not intended to be sent to the host attached to the UNID. The sequence bit is interrogated to determine its state. If the received sequence bit is the same state as the expected received sequence bit, the information frame received at the distant UNID was received correctly and in order as denoted by the matching sequence bits. If the sequence bits do not match, then an error has occurred in the transmitted information frame or the received supervisory frame and this UNID will retransmit the information frame to the distant UNID. When the received supervisory frame matches the expected supervisory frame, the sequence bit is toggled to its next state and the acknowledge variables are reinitialized.

When an information frame is received, its header is interrogated for its destination and the received sequence bit. The received information frame may go to another UNID or to an attached host. In either case, a supervisory frame is generated to the sender using the sequence bit received in the information frame header and the frame is moved to the next buffer. Note that the UNID is not responsible for end to end sequencing; the UNID is only responsible for UNID to UNID sequencing. If the sender did not receive the acknowledgement in the supervisory frame correctly, it will resend the same information frame and the UNID will acknowledge accordingly and move the received information. It is the responsibility of the transport layer software to insure that sequencing of the information is correct from the network before passing information to the attached host. Notice that if the received frame header is corrupted and the software cannot determine if the frame is a supervisory or information frame, the buffer pointer is adjusted for the next frame and the frame just received is effectively destroyed. This type of interrogation keeps corrupted frames from being sent throughout the network and to the hosts.

Figure 4-15 is the plain language pseudocode for the second part of the route$in procedure.

```
if packet has been received from network layer then
     do
     determine network destination
     if destination for channel 1 then
          do
          determine if information in transmit buffer 1
          if no info frame in transmit buffer then
               do
               build an information frame
               service local buffer pointer
               end
          end
     if destination for channel 2 then
          do
          determine if information in transmit buffer 2
          if no info frame in transmit buffer then
               do
               build an information frame
               service local buffer pointer
               end
          end
     end
```

Figure 4-15. Route$In Procedure Pseudocode (Part 2)

This section of software determines if a packet has been received for transmission into the network. The packet header is interrogated to determine its destination. The network transmit buffer is then interrogated to determine if an unacknowledged information frame is in the buffer destined for another UNID. If an unacknowledged information frame is in the network transmit buffer, then the packet is not moved into the network transmit buffer. It will be left untouched where it is until the software reinterrogates the local to network buffer. If the network transmit buffer does not contain an unacknowledged information frame, then the software builds an information frame and moves the packet and frame header into the network transmit buffer awaiting transmission to another UNID. Routing determination to either channel one or two is simply determined by the shortest path to the destination

UNID. The routing calculation is very simple based upon the fact that the UNID network architecture is a bi-directional ring.

During the review of the data link layer software from previous work, several implementation errors were detected. For example, the received frames from another UNID were first interrogated for their destination rather than their information or supervisory function. This error caused supervisory frames to be generated for received supervisory frames. Since the supervisory frames would be acknowledged at both UNIDs ad infinitum, the network would quickly become overloaded with supervisory frames and the network would cease to function. This error and others are discussed in more detail in Appendix B.

UNID II Software Development Tools

To develop the UNID II software in a reasonable length of time, certain software tools were developed. The tools were used mainly in the testing area and are discussed here as they relate to the overall software design of the UNID II software development. Detailed descriptions and use of these tools are discussed in Chapter Five and complete listings are in Appendix H.

Software was developed on the host Intel System III to simulate both the network and data link layer software. Each of these simulations allowed the operation and validation of the UNID II software on the software development system where it was relatively easy to make software changes in order to correct errors in design and implementation. The main thrust of this development was to use the proposed operational software with the ability to insert test datagrams into receive buffers to simulate the reception of a datagram from a

host. The software has a software loop from the transmit side back to the receive buffer to show the simulation of the action and response of the network. The datagrams, packets and frames were observed to be in certain buffers at certain points in the program execution, thereby validating the fact that the datagram, packets and frames were correctly routed. The received datagram, packet and frame were displayed on the development system to confirm that the data entity did, in fact, return to it's origin. Figure 4-16, similar to Figure 4-2, shows the the data structure and flow used for the network layer simulation. The indexed array names, pointer names and semaphore names are identical with those used in Figure 4-2.

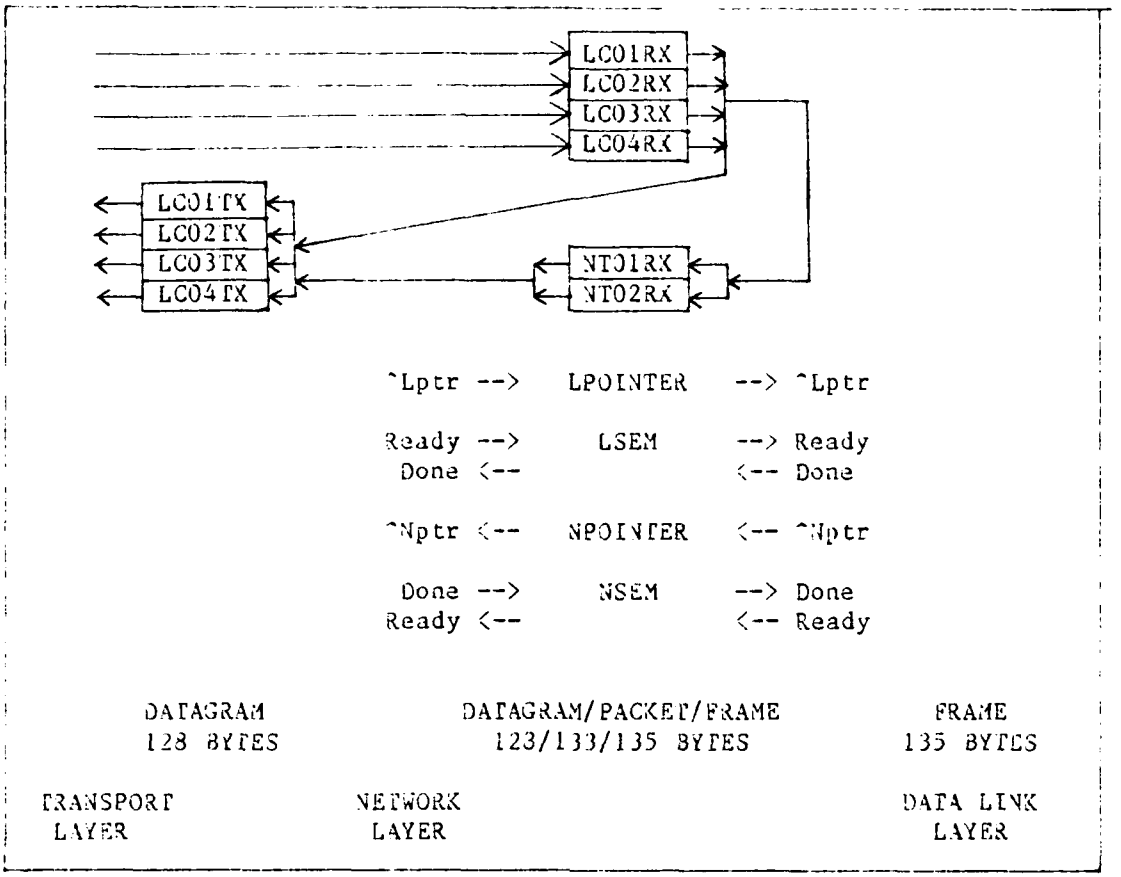


Figure 4-16. UNIX II Network Link Layer Simulation Data Structures and Flow

Figure 4-17, similar to Figure 4-1, shows the data structure and flow used for the data link layer simlulation. The indexed array names and pointer names are identical with those used in Figure 4-1.

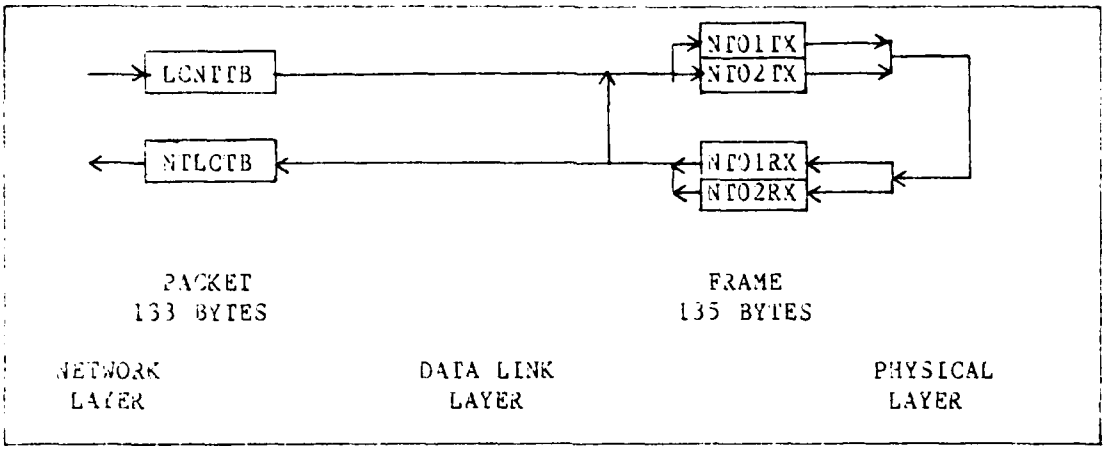


Figure 4-17. UNID II Data Link Layer Simulation Data Structures and Flow.

The development system host operating system calls were used for the operator interaction and console I/O required of the simulation. This method saved the time and effort of programming EPROMs and attempting software validation on the target hardware SBC. Once the software was validated to show that it did, indeed, route datagrams, packets and frames correctly, it was transferred to EPROMs and installed on the target SBC with the appropriate hardware initialization procedures.

Another software program was developed on the System III for use on a host processor system to transmit and receive datagrams between the host and the UNID II at the network layer. The host used for the SBC 544 interface was the a CP/M 80 system. This host was chosen because the simulation software on the CP/M system could be easily developed in PL/M80, the software could be debugged on the System III software development system, the hardware interface between the UNID II and the CP/M system existed, and last but not least, the software would execute on the CP/M system with the appropriate interface to the I/O and

operating system. This method was used with the SBC 544 board and could be used with the SBC 88/45 board simulating the UNID to UNID interface. This software validation method and system were essential to further the validation of the UNID II software. The UNID II software could then be excercised in a more realistic operational environment where the errors of design oversight and real time operation could be excercised.

A monitor program for the SBC 544 was developed from a monitor available for the SBC 86/12A board. The monitor was installed in EPROM and proved invaluable in the initial checkout and functional operation of the SBC 544 board. The paper tape read and write functions were deleted as these functions were not needed for the SBC 544. The display of memory data was modified to include an ASCII display of the memory data. A fill a block of memory with a constant function was also added. The same monitor used for the SBC 86/12A board may also be used for the SBC 88/45 board with minor hardware initialization changes.

UNID II System Memory Map

The system memory map is important because it is necessary to insure that the tables, pointers and semaphores common to both the SBC 544 and SBC 88/45 boards are correctly positioned and known to the network and data link layer software executing on each board. Each board has its own RAM, part of which is capable of being mapped into what Intel calls the system memory. System memory is that memory, with its own local board address, that is physically resident on a given SBC which is also mapped onto the multibus interface as memory available to any SBC on the same multibus. The memory address mapping on the multibus is the address that that section of RAM occupies in the system

memory. When a processor writes into a section of local memory which is also mapped into the system memory, the data which is written is also changed at the corresponding mapped address in the system memory. When a processor reads from a section of local memory which is also mapped into the system memory, the data which is read reflects the data in the locations from the corresponding mapped address in the system memory.

The system memory map used for the UNID II is depicted in Figure 4-13. All addresses are shown in the hexidecimal number base (hex). Note that the SBC 544 only has an addressing space of 0 - FFFF hex while the system memory and SBC 88/45 have address spaces of 0 - FFFFF hex.

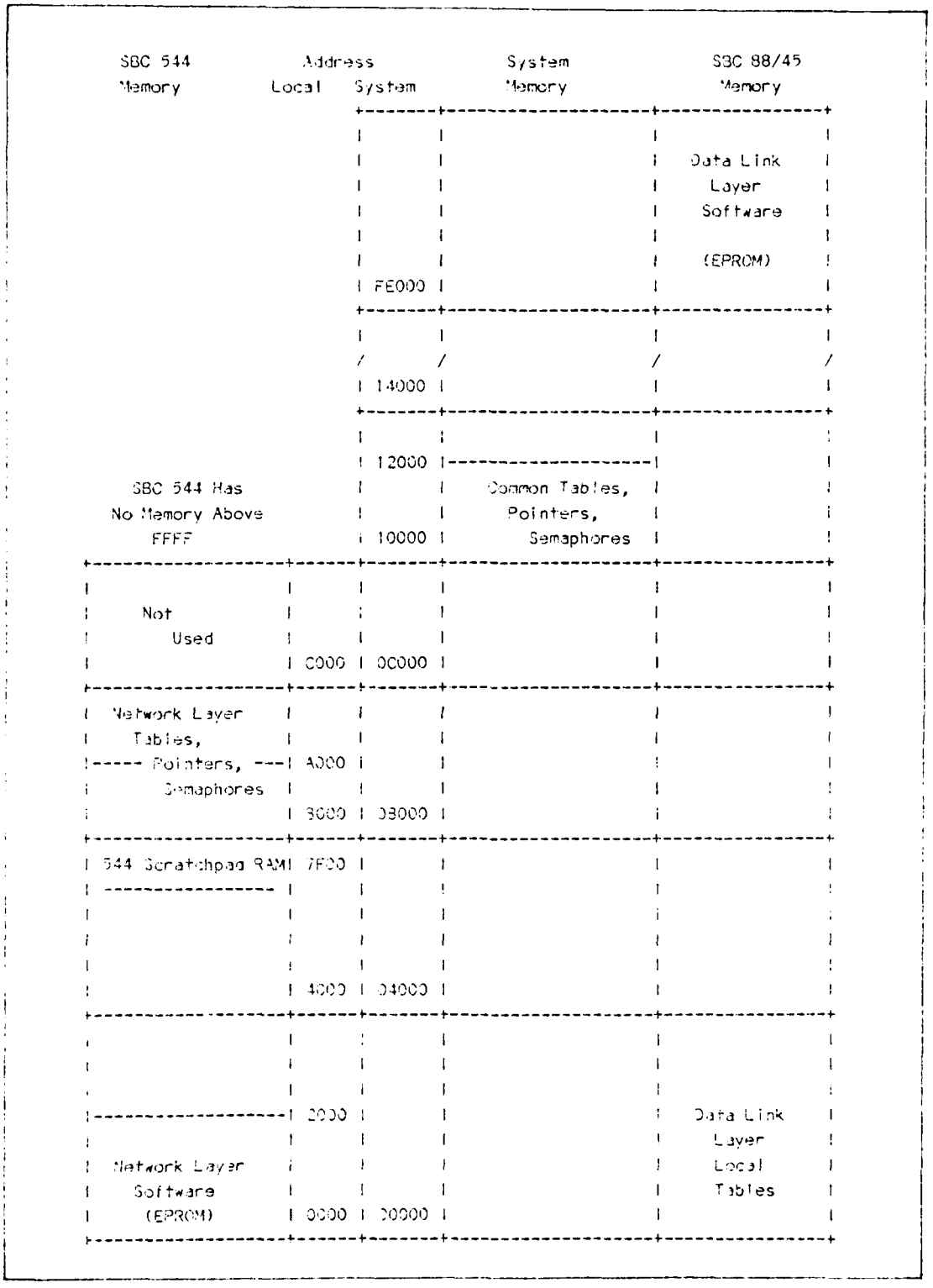


Figure 4-13. UNID II Memory Map

The common tables, pointers and semaphores are in system memory between 10000 and 11FFF hex. The SBC 544 can map 1000, 2000 or 4000 hex of its available 4000 hex memory onto the multibus at even 1000 hex boundaries (47), but only 2000 hex is required to be mapped into system memory to accommodate the common tables, pointers and semaphores. The physical RAM for the common software parameters is on the SBC 544 board and must be there because the 8085 processor space on the board cannot reach to the 10000 hex and above addresses and therefore cannot read or write above FFFF hex. The above memory mapping is also useful during the testing phase as it allows another SBC to view the SBC 544 memory contents. This feature is explained in more detail in Chapters Five and Six.

The SBC 88/45 board, however, can reach the entire system memory span as its host processor is the 8088. This arrangement allows access to shared system memory by both boards. The SBC 88/45 board can map either 3000 hex or none of its available 4000 hex memory into system memory. No SBC 88/45 board memory is mapped into system memory for this design. The SBC 88/45 board memory could be mapped into any section of system memory not otherwise used by another board on the multibus. It is recommended by this author that any SBC 88/45 board memory mapping be above 10000 hex. This allows another SBC, such as the SBC 86/12A or 86/30, to view the memory of the SBC 88/45. This feature is extremly useful for troubleshooting problems during the testing phase, as further explained in Chapters Five and Six.

The EPROM for the SBC 544 resides in the local memory at 0 - 1FFF hex. The 2000 hex size is the maximum for that board using two 2732A EPROMs. The EPROM for the SBC 88/45 board is at FC000 - FFFFF hex. The

4000 hex size, while not the maximum of 8000 hex, is adequate for this application. The board uses four 2764 EPROMs.

Conclusion

This chapter discussed the detailed design and implementation of the UNID II software. The discussion was divided into six sections which included: Development Language Selection; UNID II Data Structures; UNID II Network Layer Software Design; UNID II Data Link Layer Software Design; UNID II Software Development Tools; and UNID II System Memory Map. Chapter Five discusses the UNID II testing philosophy and design.

V. Test Philosophy and Design

Introduction

This chapter outlines the test philosophy and design of the UNIO II software and hardware developed in this research and development effort. The basic test philosophy is discussed first, followed by a description of the overall test design. The major diagnostic test tools used for testing are then discussed followed by a description of each of the testing phases. Testing results are included in the discussion of each phase of testing.

Test Philosophy

The overall philosophy of most any test is to `prove', by some specified means and tools, that a certain specified event does, does not or to some degree occurs. The word `prove' may take on many meanings to many people. While some desire to see a detailed mathematical tretise to `prove' a hypothesis, others are satisified with a more pragmatic or economic approach of observing a demonstration as a means of `proof'. In some definitions, the formal mathematical proof is called verification while the more economic demonstration is called validation. Validation also provides the developer and user with some level of confidence that the item under test functions as desired. This effort uses the validation approach because of the cheaper economics to demonstrate the system works as described compared with a detailed mathematical proof that the system works as described.

While most design efforts begin with a top-down approach to the particular problem using step-wise refinement techniques to finally achieve a tractable solution, a considerable amount of testing effort

begins at the bottom and works its way to the top. The testing of the UNID II hardware and software takes the bottom-up approach. The approach follows that often used with software testing, specifically: module testing, integration testing, and overall systems testing. While there are many testing methodologies, such as static and dynamic testing, applied to three phases of testing, the general methodology used in this effort, in following with previous testing efforts (75:3-1), will be path testing. Boundary condition testing, while important in its own right, was not emphasized during the testing. Some boundary testing and provision for display of the boundary violations is provided in the software design itself and available to the tester.

Overall Test Design

Since the ultimate goal of the UNID II is to provide the communications media for host computer systems, the testing is oriented towards the validation of host-to-host communications through the UNID. To that end, UNID II testing was divided into five phases. The testing began at the lower level software modules and progressed to the larger programs which integrated the lower level modules. Both of these phases used the Intel System III and simulation as the test vehicle. The third phase was involved with the interface and preliminary system test of the SBC 544 with a simulated host on a CP/M system. This phase involved the use of the operational software on the SBC 544 and simulation software on the CP/M host. The fourth phase was a system test of the SBC 544 with the LSI-11 NETOS. The SBC 544 with operational software, the host CP/M system, a display terminal, and four of the nodes in the LSI-11 NETOS were the test vehicles. The fifth and last

phase was to connect two or more UNID IIs in their operational network configuration and attach host computer systems to the UNIDs and conduct host-to-host communications through the UNID network.

Only the first four phases were conducted. The fifth phase was not conducted because only one UNID II was available for development and test. As more UNID II hardware is acquired, this last phase is the next logical step in the development, test and operation of the UNID II in the DELNET.

Major Diagnostic Test Tools

While the test philosophy, design and methodology outline the major steps to determine if an entity is operating properly, the test tool is the basic instrument used to implement the actual testing. Path testing is the major methodology used in the UNID II testing to determine if the software was operating properly. Path testing essentially shows that the entity under test follows a desired path while traversing some media from its source to its destination. To implement the path testing methodology, three major diagnostic tools were developed and used.

The major diagnostic tool used for path testing in this, and previous work (64, 75), is to insert diagnostic messages at strategic locations in the various software modules to show that a process or the data did, in fact, follow a certain path. The diagnostic messages are typically displayed on a computer terminal attached to the hardware under test or at the terminal of a development system, as dictated by the particular test. When a message is displayed when it should be displayed, the observer is assured that the software under test did, in fact, follow the desired path. If a message is displayed when it should

not be displayed, the observer is assured that the software traversed an incorrect path. If a message is not displayed when it should be displayed, the observer could conclude that a problem exists in the hardware, software or their own perception of when the message should be displayed. The lack of a displayed message is not conclusive proof that a problem exists in the software as other conditions may exist to prevent the display of the message. The observer needs to look further at the conditions of the test, or even generate other tests or diagnostics, to determine if the software is at fault.

Another major diagnostic tool used in the UNID II testing was to use the monitor program on SBC 86/12A with the SBC 544 to view the data tables, variables, pointers, flags and semaphores in the SBC 544 memory. As the reader will recall from Chapter Four, the memory of the SBC 544 was mapped into system memory for two reasons: to pass data, pointers and semaphores to the SBC 88/45 and to allow an external processor to view the memory contents of the SBC 544 during operation. This tool was used where it was not feasible to insert diagnostic messages, such as attempting to display the contents of a large number of variables or data, as well as to view the memory contents during the operation of the UNID II. The ability to observe the contents of the variables proved to be invaluable while attempting to trouble shoot software that did not work as designed.

The last major diagnostic tool was a CP/M system used to simulate a host to the UNID II. This tool's advantage lies with the ability to send and receive datagrams from the UNID II. This ability allowed the UNID II to be tested in a more real time environment and provide a more realistic test of the UNID II software. This tool was chosen because the

system was readily available where others were not, the simulation software was relatively easy to develop, the RS-232 hardware interface to the UNID II existed.

Phase One Testing

Only a few of the low level software modules from the original software (64, 75) were tested on a module basis. The tested modules were related to the display of the diagnostic path testing messages on the host operating system or other display terminal. A small print program was developed to validate the correct interface with the ISIS operating system calls. Messages were typed at the operator keyboard and then displayed on the ISIS terminal. The remaining modules had been previously tested (75:Ch 3, Ch 5) and were assumed to be correct with the exception of implementation errors during the software conversion from PL/Z to PL/M.

Phase Two Testing

A simulation capability was developed on the software development the Intel System III, which allowed the development and test of the individual modules through the fully integrated, single board computer level programs. The simulation method allowed the functional validation and testing of the fully operational software on the development system. When the validated software was then installed on the single board computer, it was known to correctly receive, interpret datagram, packet and frame headers, and route the data correctly since it had just been validated via simulation with a large degree of confidence that the software functioned properly.

This simulation capability was decided upon because it was

'simpler' and more time efficient to test the software functionality before installing the software on the single board computers and testing with the in-circuit emulator (ICE), the ICE 86A, the CP/M system, or the NEIOS. There was also no ICE 85 (40) available for the SBC 544 board during the testing period. In addition, the ICE is used mainly for new designs where the hardware is designed and developed from scratch and does not use off the shelf single board computers. The simulation capability allowed the testing of software modules where incorrect software functioning could be readily detected and corrected without the necessity of executing the unproved software on the target hardware system. Experience with the systems showed that software changes could be readily made, the program recompiled and retested in approximately 10 to 20 minutes where similar changes for the target hardware required 30 or more minutes to complete.

The interface of the programs with the System III required the use of the ISIS operating system calls for message display and operator keyboard interaction. Appropriate software procedures were developed to insert datagrams and packets in the buffers of the operating software and to display the contents of the buffers. Figures 5-1 and 5-2 show the operating software data flow and the software loops. The labels are the same used for Figures 4-1 and 4-2.

In Figure 5-1, datagrams were inserted in the LC01RX receive buffer with operator options allowing the choice of destination and number of datagrams. The destination was chosen by selecting a UNID number. The operator then had the option of choosing the destination port by selecting a host code between 0 and 255. The network layer software then routed the datagram through the tables and displayed the received

datagram on the simulation terminal. Test point messages displayed to the System III terminal were embedded in the software in each procedure, and in some cases, within the if-then-else and case statements to validate the execution of a particular section of code.

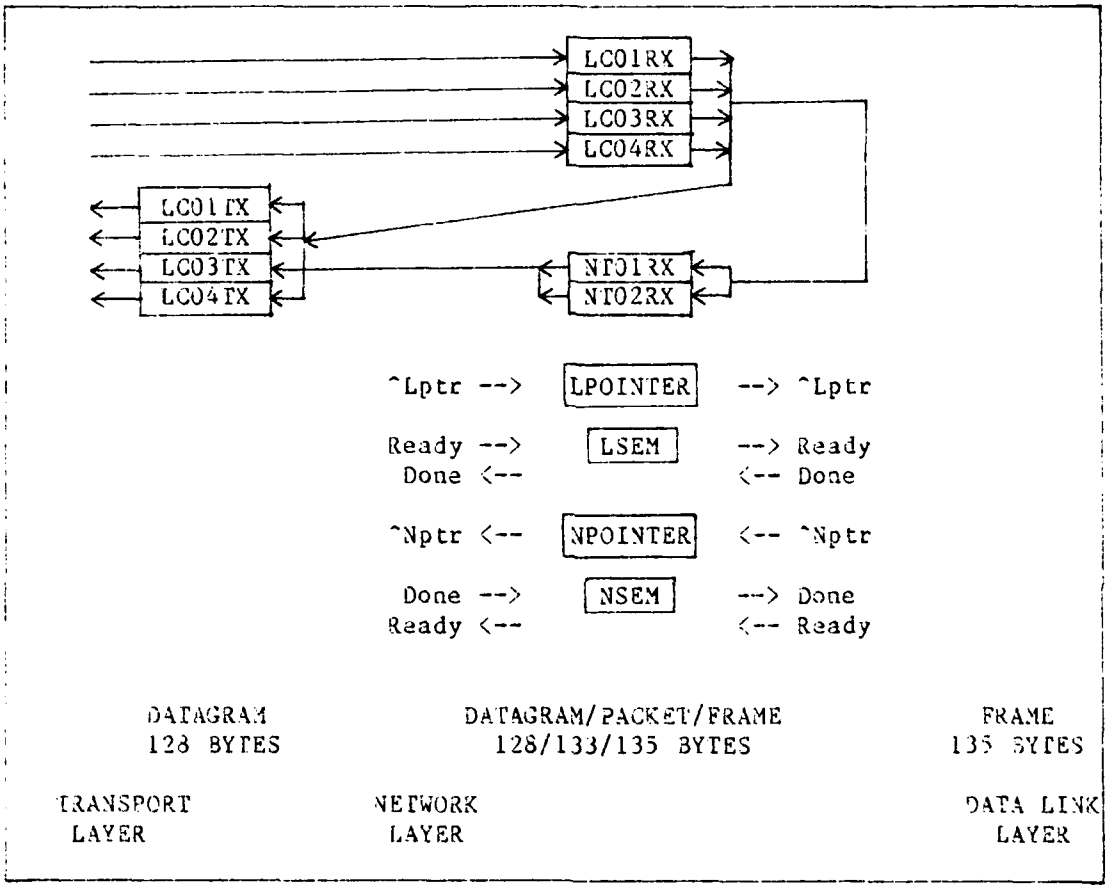


Figure 5-1. Network Layer Simulation Data Structure and Flow

The test datagrams were simple messages with a modulo 10 datagram counter imbedded in the datagram. Test messages were inserted at certain locations where an out of range error would occur to advise the operator that an error had occurred. This is an example where boundary testing was designed into the path testing messages. The IP header was also displayed for troubleshooting and validating that the software did

manipulate the header as and when it should. A software loop from the transmit tables to the receive tables simulated communication with another UNID through the data link layer and DELNET. Datagrams were inserted in each of the four receive buffers with each of the transmit buffers as their destination. This test was successfully accomplished both in the local-to-local host communication and local-to-distant host communication modes.

The same type of testing was used with the data link layer software shown in Figure 5-2.

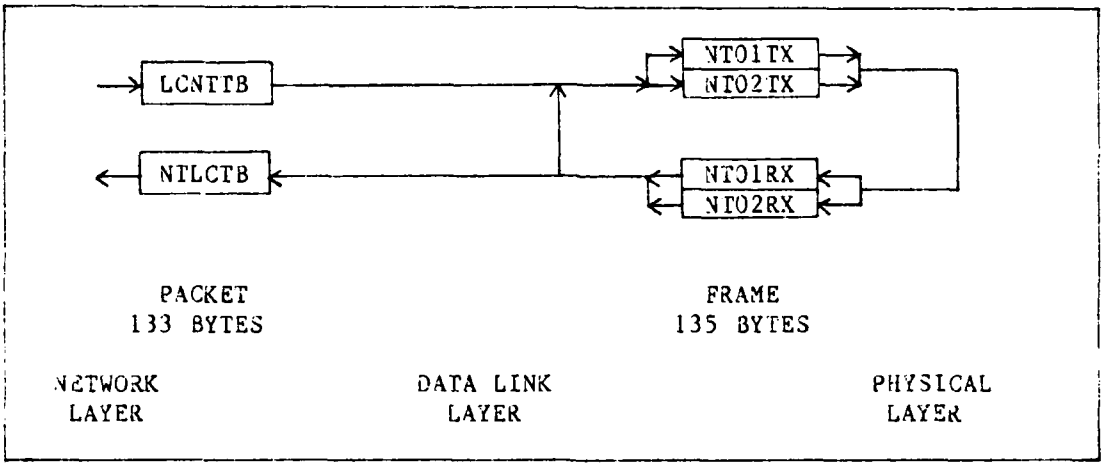


Figure 5-2. Data Link Layer Simulation Data Structure and Flow.

Packets were inserted in the LCNTTB buffer with operator selected destinations and number of packets to send. The receive test packets were read from the NTLCTB and displayed with their modulo 10 counter to the operator on the System III terminal. In this test, the test messages were also inserted in various strategic locations in the software as with the network layer software. Test messages were also inserted at certain locations where an out of range error would occur to advise the operator that an error had occurred. This is another example

where boundary testing was designed into the path testing messages. The packet and frame headers were not displayed as there was no need to do so. The test messages showing the flow of the datagram sufficed to validate the software. A software loop from the transmit tables to the receive tables simulated communication with other UNIDs through the DELNET. This test was successfully accomplished through all the tables.

The simulations on the System III were conducted with the actual operational software that would execute on the SBC 544 and SBC 88/45 single board computers. As errors were found in the design or logic of the original software, appropriate modules were corrected or rewritten and tested both on a module by module and overall program basis. The implementation errors from previous work are discussed in Appendix C.

Phase Three Testing

Phase three testing consisted of testing the network layer software on the SBC 544 with a simulated host on a CP/M system. As previously mentioned, the data link layer software was not tested on its host SBC 88/45. The simulation software was modified to delete the ISIS operating system calls, install the procedure that displayed messages on an Heath H19 terminal attached to the SBC 544, and install the procedures to initialize the special purpose large scale integrated circuits on the SBC 544. Once these steps were accomplished, the software was compiled and programmed into EPROMs for the SBC 544. Programming the EPROMs involved the use of one of two EPROM programmers for the two different EPROMs used. An Intel EPROM programmer (60) was attached to a second Intel system, a System 210. A software program on

the System 210 was used to program 2716 EPROMs. A Bytek EPROM programmer (8) was attached to the CP/M system to program 2732A EPROMs. A communications program on the CP/M system downloaded the software to the intellegent programmer to program the 2732A EPROMs. Software transportability between the System III and the System 210 was accomplished with single sided, single density ISIS formated diskettes. Both Intel systems can read and write single sided, single density diskettes. The above procedures were developed for this test as there were no prior established procedures.

Two different size EPROMs were used because the initial testing in this phase used only one receive and one transmit table for the network layer software and did not require a large amount of memory, the interrupt handling procedures for the SBC 544 needed to be developed and tested, and the 2716 EPROM programmer was directly available. This approach was taken until the interrupt handling procedures worked properly. In addition, the appropriate compiler constructs for the PL/M compiler needed to be validated in order to correctly locate the receive and transmit tables in the SBC 544 memory. Then the software was expanded to its full size, compiled and installed in the 2732A EPROMs.

Transporting the software from the System III to the CP/M system required an intermediate translation process on the System 210 as the software was on an ISIS formatted diskette on the System III and needed to be in a standard CP/M format for the CP//M system. A commercially available program, procured by a previous researcher (64), was available to run on the System 210 under the CP/M operating system. The program is a file format translation program which converts files from ISIS to CP/M or CP/M to ISIS formatted diskettes. After the network layer

software was translated, it was then downloaded to the EPROM programmer from the CP/M system. The EPROMs were then installed on the SBC 544 and the program executed.

The simulated host software that executed on the CP/M system was also developed on the System III. The software was translated on the System 210 as explained above and transported to the CP/M system where it was executed. The software used many of the procedures already developed for the network and data link layer simulations. The additional software required consisted of interfacing the operator query and response to the CP/M console through CP/M operating system calls. The serial communication interface between the SBC 544 and the CP/M system was a standard RS-232C cable with USART I/O driver software on the CP/M system provided by an assembly language program for that purpose. The assembly language program was already available (Appendix 1) and only needed slight modification to interface with the PL/M procedure calling conventions and add the CP/M operating system call interface for the main PL/M program. The register usage of the PL/M procedure calling conventions are identical to many of the CP/M operating system calls and allowed a smooth interface.

The simulated host software was able to send and receive up to ten datagrams at one time as specified by the operator. The operator could also specify the routing of the datagram as in the network layer simulation. The testing then consisted of sending datagrams from the simulated host to the SBC 544, observing the diagnostic messages displayed on the H19 terminal and observing the response from the simulated host. When incorrect or unexpected software behavior occurred, the test messages were used in conjunction with the software listings to

trace the flow of the program through its execution. In addition, the use of the SBC 86/12A and its monitor were used to observe table, counter and boolean flag contents to aid the determination of where the software was not functioning correctly. The H19 was connected to only one port during the test for simplicity. The RS-232C cable was connected sequentially on the three remaining ports and datagrams sent from the simulated host to the SBC 544 and returned on each of the three SBC 544 ports. Both a simple local-to-local port loop and as well as a local-to-distant host loop were tested successfully for all three ports. The transmit request/transmit acknowledge handshake, discussed in Chapter Four and Appendix F, was also validated. Figure 5-3, similar to Figure 5-1, shows the network layer data flow and structures with the addition of the H19 monitor and the CP/M system used in this phase of testing.

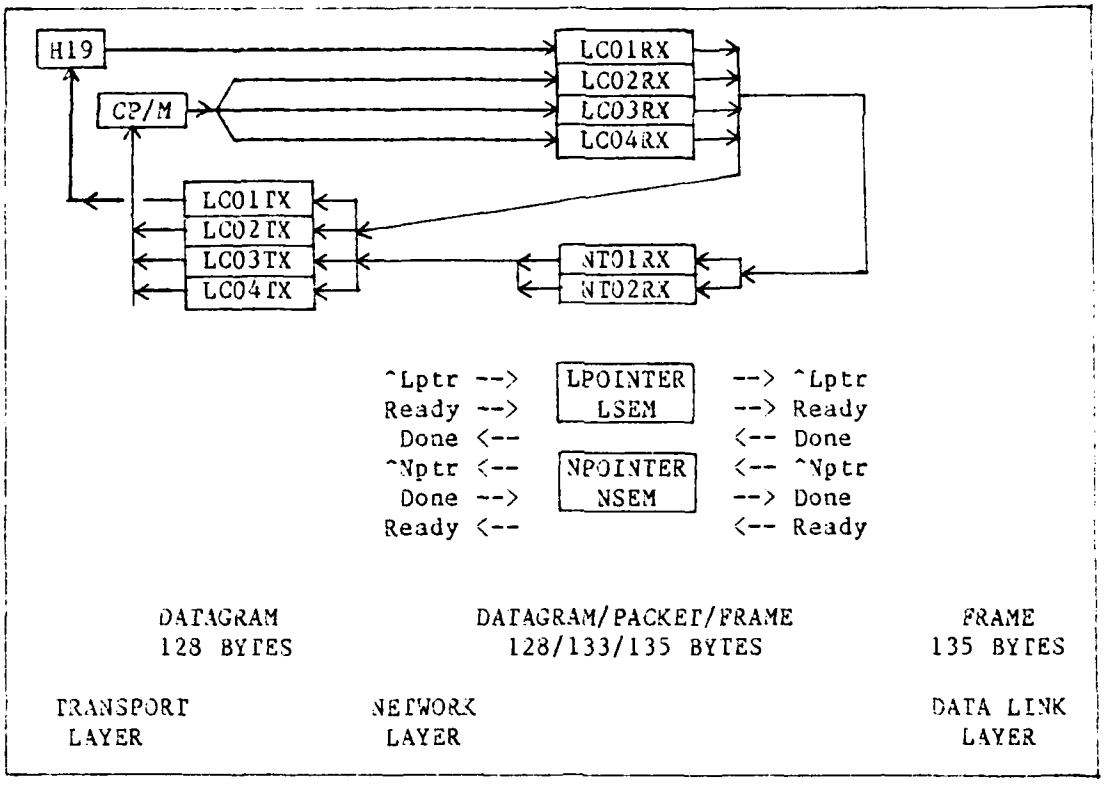


Figure 5-3. Network Layer Simulation with the CP/M System and H19

This phase of testing was particularly significant for two reasons. First, both the receive and transmit side of each of the SBC 544 serial ports are interrupt driven. This is significant because the software had to behave rationally while transmitting and receiving random datagrams on a real time interrupt basis while `simultaneously' routing datagrams. Second, it was quickly discovered by observing incorrectly behaving software that certain sections of the executing software needed to be protected from outside interference while they were executing. These sections of code are called critical regions (13:77) and are discussed in Chapter Four and Appendix E. This phase of testing also allowed the testing to occur under more real conditions than the simulations and demonstrated the effects of real time interrupt

programming and critical regions that were not possible during the simulations on the System III development system.

Phase Four Testing

This phase of testing consisted of testing the network layer software from the phase three testing on the SBC 544 while the SBC 544 was connected to the LSI-11 NETOS (72). Figure 5-4 shows the connection of the UNID II with the NETOS.

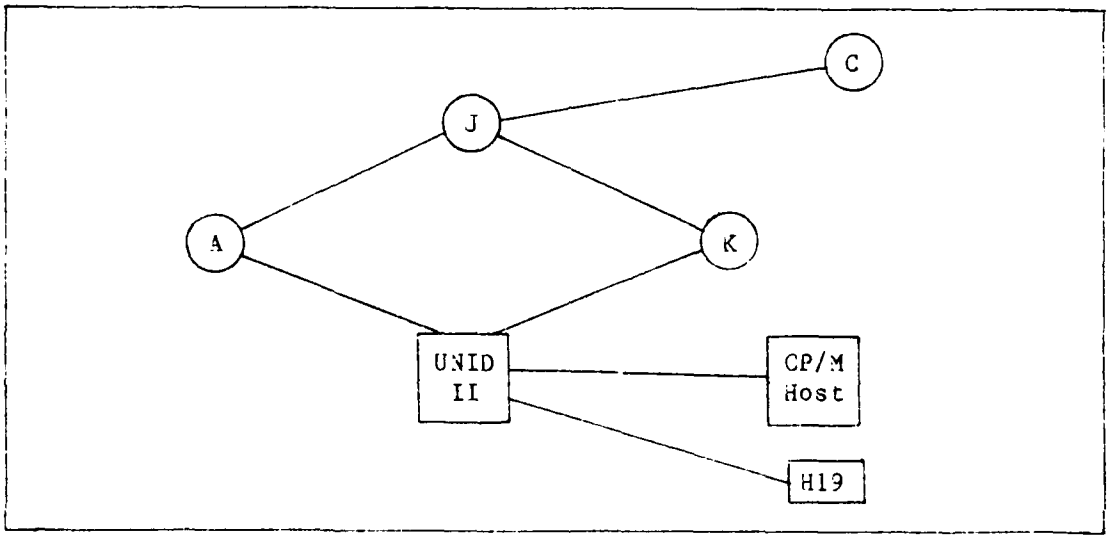


Figure 5-4. UNID II and NETOS Connection

The letters A, C, J, and K are the designations for the NETOS nodes used for this test. Each NETOS is a LSI-11 processor executing the NETOS network software (72). The software executing on the NETOS was developed as part of a class at the Air Force Institute of Technology. The software implemented a half-gateway (87:Chap 3) interfacing the NETOS layer four protocol with the UNID II. The UNID II half-gateway is implemented in the network layer software that interprets the Internet Protocol (IP) header. Both the NETOS and the UNID II used the IP header for internet datagram routing. Node J is the central node through which

all the NETOS nodes are connected in a star configuration. Node C was used to insert NETOS messages into the NETOS destined for the UNID II. Nodes A and K were connected to two of the UNID II ports while the CP/M host and the H19 were connected to the two remaining UNID ports. The CP/M host and the H19 terminal were used as in the phase three testing while node C was inserting NETOS messages into the NETOS and through the UNID.

Local-to-local routing was tested where messages inserted in the NETOS circulated clockwise and then counter clockwise in the ring A-J-K-UNID-A in Figure 5-4. This test validated the local-to-local routing process in the UNID network layer software. Other messages were injected at node C to address another node hosted on another UNID. This test simulated one NETOS node sending messages through two UNIDs to another NETOS node and receiving replies by the same path. In this test, the messages traversed the path C-J-K-UNID-UNID-A-J-C. The UNID-UNID designation represents the software loop in the network layer software. This software loop remained in the network layer software for this test as a second UNID II was not available.

Messages were also inserted at the CP/M system while the NETOS nodes A and K were sending and receiving messages with the UNID. This test was done to provide a somewhat higher message loading on the UNID to subjectively observe any detremental effects. There were no detremental effects to the NETOS messages by the insertion of additional messages by the CP/M system during the subjective observation. This phase of testing validated the local-to-network an network-to-local routing process in the UNID network layer software.

Of particular note during this test were instances where the UNID

software `lost' data from the sending node. The lost data was a result of the code in the UNID required to protect a critical region and of the code in the NETOS which sent the NETOS messages. The NETOS messages were broken into two datagrams which were sent to the UNID in rapid succession. The UNID successfully received the first datagram from node K and while attempting to send that datagram to node A, the second datagram began arriving from node K. The lost data occurred during the reception of the second datagram from node K. The cause of the lost data was the result of the disabled interrupts required to protect a critical region in the UNID software. Data was continued to be sent by node K, but while the UNID interrupts were disabled during transmission of the first datagram to node A, some of the data from the second datagram simply overflowed in the UNID USART and were lost. The fact that bytes were in fact being lost was determined by using the SBC 86/12A monitor to check the program variables in the SBC 544 memory. It was quickly observed by checking the received byte count that only 125 bytes of the 128 byte datagram were received in the SBC 544 memory. A `quick fix' patch was made to the NETOS software to increase a delay between transmission of characters to the UNID. The patch was made to the NETOS software instead of the UNID software as the UNID software had to be reinstalled in EPROMs before retesting. The additional transmission delay between characters from the NETOS to the UNID alleviated the problem during the testing. The solution lies within the UNID software and has not yet been established.

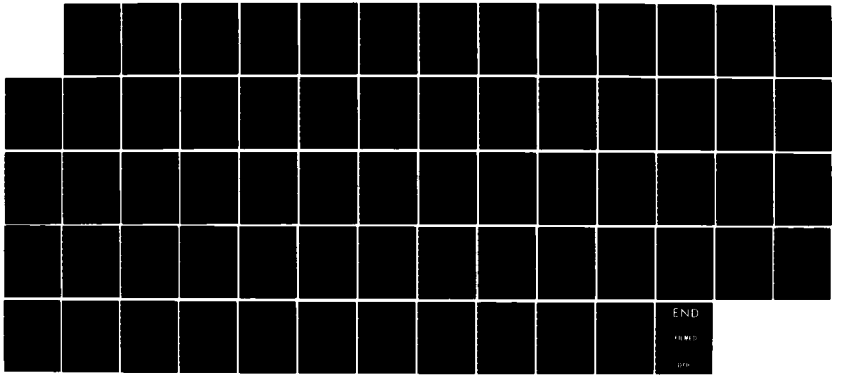
Conclusion

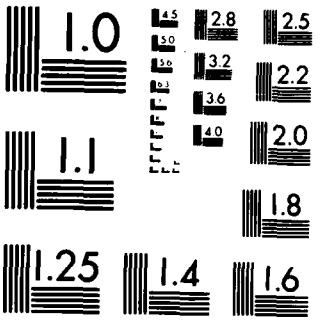
This chapter outlined the test philosophy and design of the UNID II

AD-A151 935 CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE UNIVERSAL NETWORK INTERFA.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C T CHILDRESS

2/2

UNCLASSIFIED DEC 84 AFIT/GE/ENG/84D-17-VOL-1 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

software and hardware developed in this effort. The basic test philosophy was discussed first, followed by a description of the overall test design. The major diagnostic test tools used for testing were then discussed followed by a description of each of the testing phases. Testing results were included in the discussion of each phase of testing. The first four of five phases of testing were completed. The next chapter discusses the conclusions of this thesis effort and recommendations for further research and development.

VI. Conclusions and Recommendations

Introduction

The purpose of this investigation was to develop the UNID II and implement the existing software designs to produce a functional unit. Well over the majority of the tasks in the problem statement in Chapter One were accomplished as described in Table VI-I. The integration of the network layer and data link layer software on the SBC 544 and SBC 88/45 and test and validation as a functional UNID II was not accomplished.

Table VI-I. Tasks Accomplished.

1.	A software monitor for the SBC 544 was developed.
2.	The SBC 86/12A and SBC 544 were integrated for operation on a multibus chassis.
3.	The functional operation of the translated network layer PL/M software was validated by simulation and test.
4.	The integration, test and validation of the network layer software hosted on the SBC 544 was completed.
5.	The data link layer PL/Z software (75) was translated to PL/M and validated for correct functional operation by simulation on a software development system.
6.	The preliminary integration, test and validation of the network layer software and SBC 544 in the DELNET was accomplished.

Conclusions

The development of the UNID II in the DELNET has progressed considerably. The use of functional simulations on a host software development system has enabled the developer to validate the software before installation and test on the target hardware system. This

capability can save considerable time and effort attempting to troubleshoot and problem solve on a target system by eliminating most of the errors in the software design and code before the software is executed on the hardware.

The integration of the network layer software hosted on the SBC 544 with the NETOS is a major milestone not achieved before this thesis effort. Two different hardware systems executing software designed and implemented by two different groups of people successfully transferred information. Although work remains to be done to fully implement the UNID in the DELNET, a solid foundation has been developed which should make the task less formidable.

Recommendations

As this investigation is a continuation of previous research and development, the recommendation by *this* author is to continue the UNID II and DELNET development. This project has provided a test bed for some of the practical experience required in applying the theory learned in the classroom. Specific examples from this effort are the real time interrupt programming and critical regions of software. Both of the examples are discussed in the classroom environment, however one does not gain the full impact of the theory and what needs to be done to properly design the software until one actually designs, tests and validates software of that type.

Not only has this project provided valuable practical experience, but the U.S. Air Force has a growing need for a UNID. The Air Force and other DoD departments are incorporating a considerable number of single and multiple user microprocessor based systems in the office

environment. While many of these systems have some common media for transferring information, many do not. The UNID can more than adequately fill the gap in linking these systems together to transfer their information. If for no other reason, the UNID development should continue to fill that gap.

Other specific recommendations not yet addressed are:

1. The analysis of the buffer sizes, throughput, delay, and host and network serial data rates has not yet been accomplished since the UNID development began in 1978. This analysis is required and must eventually be accomplished in order for the UNID to adequately support the user data traffic. The methods outlined in (87:Chap 2, 81, 82) could be used as a starting point for the analysis. This task could develop into an entire thesis effort.

2. Only the first four of the five test phases were conducted in this effort. The fifth test phase of integrating the network layer and data link layer software on the SBC 544 and SBC 88/45 and validating as a functional UNID II in the DELNET remains to be accomplished. This test and validation can be accomplished as more UNID II hardware is acquired. This last phase is the next logical step in the development, test and operation of the UNID II in the DELNET and is required to be done.

3. The timing problem (Chap Five) which arose during the testing of the UNID II and the NETOS needs to be resolved.

4. The data link layer software needs to be installed on the SBC 88/45 single board computer and validated in a network environment.

5. The further implementation of the X.25 network layer protocol remains to be accomplished, as well as the implementation of the higher

layer protocols for UNID hosts computers and the internet protocol (IP) for networks connecting to the UNID. These implementations must occur for a functional DELNET to exist.

6. Use the C language for further software development, especially in the transport through the presentation layers of the OSI model.

7. The DELNET and the supporting UNIDs would comprise an excellent tool for research and development of user data privacy, user authentication, and user `verification' implementations. While classified traffic cannot be supported in the AFIT environment, the basic groundwork for a classified system can be supported. The DoD has a considerable interest in this area (2) and much can be gained. Some previous investigation at AFIT has been started (86) with a considerable amount in the commercial sector (5, 10, 14, 15, 65, 79, 83). (10) and (83) provide introductory material while (5, 14, 15, 65, 79) provide more advanced material. Research and development in these areas can easily encompass several theses.

Concluding Remarks

A considerable amount of time, effort and resources have been applied to the design, implementation and testing of the UNID and the DELNET. There is, however, still much work to be done before the UNIDs can perform in a local area network function. It is this author's opinion that this project is valuable and can pay large dividends, both to satisfy U.S. Air Force operational needs and as a learning tool, when continued to fruition.

Bibliography

1. 1842 EEG/EEIC. An Engineering Assessment Toward Economic, Feasible and Responsive Base Level Communications Through the 1980's. Technical Report TR 78-5. Richards-Gebaur AFB, Missouri. October 1977.

2. Air Force Automated Systems Program Office (AFASPO) (AFCC). Private conversations with the program management staff regarding multi-user and multi-level security issues related to the Integrated-Service/Agency Automated Message Processing Exchange (I-S/A AMPE). Gunter AFS, AL, June-July 1982 and March-April 1983.

3. Arthurs, Edward, Gregory L. Chesson and Barton W. Stuck. "Theoretical Performance Analysis of Sliding Window Flow Control," IEEE Journal on Selected Areas in Communications, SAC-1: 947-959 (November 1983).

4. Baker, L. "USAF Prototype and Software Development for Universal Network Interface Device." MS thesis, AFIT-GCS-EE-80D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1980 (AD A100787).

5. Beth, Thomas. Cryptography. Lecture Notes in Computer Science, number 149. Berlin and Heidelberg West Germany: Springer-Verlag, 1983.

6. Borgsmiller, Michael. "The Serial Communications Interface Board." Project Report, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1983.

7. Brown, E. "USAF Prototype Universal Network Interface Device." MS thesis, AFIT-GE-EE-79-8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1979 (AD A080173).

8. Bytek Corporation. EPROM Programmer Manual. Manufacture's data. Santa Clara CA, 1982.

9. Cole, Kenneth. Private conversations regarding data structures. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June - September 1984.

10. Cooper, James Arlin. Computer Security Technology. Lexington Massachusetts: D.C. Heath and Co., 1984.

11. Cuomo, Gennaro. "Continued Development of the Universal Network Interface Device." MS thesis, AFIT-GE-EE-82D-28. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.

12. Day, John. D. and Hubert Zimmermann. "The OSI Reference Model," Proceedings of the IEEE, 71: 1334-1340 (December 1983).

13. Deitel, Harvey M. An Introduction to Operating Systems. Reading Massachusetts: Addison-Wesley, 1984.

14. Denning, Dorothy E. Cryptography and Data Security. Reading Massachusetts: Addison-Wesley, 1982.

15. Dinardo, C. T. Computers and Security. The Information Technology Series, Volume III. Montvale NJ: American Federation of Information Processing Societies, Inc., 1978.

16. DOD Standard: Transmission Control Protocol Specification. Arlington VA: Defense Advanced Research Projects Agency (DARPA), September 1981.

17. DOD Standard: Internet Protocol Specification. Arlington VA: Defense Advanced Research Projects Agency (DARPA), September 1981.

18. EIA Standard RS-232C. Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, April 1975.

19. EIA Standard RS-449. General Purpose 37 Position Interface for Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, November 1977.

20. Fairchild Camera and Instrument Corporation. Microprocessor Products Data Book. Manufacturer's data. Santa Clara, California: Fairchild Camera and Instrument Corporation, January 1983.

21. Specification for Message Format for Computer Based Message Systems. Proposed Federal Information Processing Standard. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, April 1982.

22. Specification of a Transport Protocol for Computer Comunications, Volume 1: Overview and Services. Draft Report ICST/HLNP-83-1. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, January 1983.

23. Specification of a Transport Protocol for Computer Comunications, Volume 2: Class 2 Protocol. Draft Report ICST/HLNP-83-2. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, February 1983.

24. Specification of a Transport Protocol for Computer Comunications, Volume 3: Class 4 Protocol. Draft Report ICST/HLNP-83-3. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, February 1983.

25. Specification of a Transport Protocol for Computer Comunications, Volume 4: Service Specifications. Draft Report ICST/HLNP-83-4. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, January 1983.

26. Specification of a Transport Protocol for Computer Comunications, Volume 5: Guidance for the Implementor. Draft Report ICST/HLNP-83-5. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, January 1983.

27. Specification of a Transport Protocol for Computer Comunications, Volume 6: Guidance for Implementation Selection. Draft Report ICST/HLNP-83-6. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, February 1983.

28. Folts, Harold C. Revised CCITT Recommendation X.25, 1980. NCS TIB 80-5. Washington D.C.: National Communications System, Office of Technology and Standards, August 1980 (AD A092394).

29. Geist, John W. "Development of the Digital Engineering Laboratory Computer Network: Host-to-Node/Host-to-Host Protocols." MS thesis, AFIT-GCS-EE-81D-8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1981.

30. Gravin, Andrew G. "Preliminary Design of a Computer Communications Network Interface Device Using INTEL 8086 and 8089 16-Bit Microprocessors." MS thesis, AFIT-GCS-EE-81D-9. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1981.

31. Hartrum, Thomas C. Private conversations regarding the AFIT LSI-11 Network Operating System (NETOS). School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June - September 1984.

32. Hazelton, Craig H. "Continued Development and Implementation of the Protocols for the Digital Engineering Laboratory Network." MS thesis, AFIT-GE-EE-82D-37. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.

33. Hobart, William C., Jr. "Design of a Local Computer Network for the Air Force Institute of Technology Digital Engineering Laboratory." MS thesis, AFIT-GE-EE-81M-3. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1981.

34. Intel Corp. Alter Text Editor User's Guide. Manufacturer's data, 121956-001. Intel Corp., Santa Clara CA, 1982.

35. Intel Corp. Component Data Catalog. Manufacturer's data, 210298-001. Intel Corp., Santa Clara CA, 1982.

36. Intel Corp. Development Systems Handbook. Manufacturer's data, 210940-001. Intel Corp., Santa Clara CA, May 1983.

37. Intel Corp. Distributed Control Modules Databook. Manufacturer's data, 230972-001. Intel Corp., Santa Clara CA, 1984.

38. Intel Corp. iAPX 86, 88 Family Utilities User's Guide. Manufacturer's data, 121616-003. Intel Corp., Santa Clara CA, 1982.

39. Intel Corp. iAPX 86/88, 186/188 User's Manual: Programmer's Reference. Manufacturer's data, 210911-001. Intel Corp., Santa Clara CA, May 1983.

40. Intel Corp. ICE-85B In-Circuit Emulator Operating Instructions for ISIS-II Users. Manufacturer's data, 980463-003. Intel Corp., Santa Clara CA, 1982.

41. Intel Corp. ICE-86A/ICE-88A Microsystems In-Circuit Emulator Operating Instructions for ISIS-II Users. Manufacturer's data, 162554-002. Intel Corp., Santa Clara CA, 1982.

42. Intel Corp. Intellec Series III Microcomputer Development System Programmer's Reference Manual. Manufacturer's data, 121618-003. Intel Corp., Santa Clara CA, 1981.

43. Intel Corp. Intellec Series III Microcomputer Development System Product Overview. Manufacturer's data, 121575-002, 1981. Intel Corp., Santa Clara CA, 1981.

44. Intel Corp. Intellec Series III Microcomputer Development System Console Operating Instructions. Manufacturer's data, 121609-003. Intel Corp., Santa Clara CA, 1981.

45. Intel Corp. iSBC 86/12A Single Board Computer Hardware Reference Manual. Manufacturer's data, 983074-01. Intel Corp., Santa Clara CA, undated.

46. Intel Corp. iSBC 88/45 Advanced Data Communications Processor Board Hardware Reference Manual. Manufacturer's data, 143824-001. Intel Corp., Santa Clara CA, 1983.

47. Intel Corp. iSBC 544 Intellegent Communications Controller Board Hardware Reference Manual. Manufacturer's data, 9806168. Intel Corp., Santa Clara CA, 1983.

48. Intel Corp. ISIS-II User's Guide. Manufacturer's data, 980306-05. Intel Corp., Santa Clara CA, 1979.

49. Intel Corp. ISIS-II PL/M-80 Compiler Operator's Manual. Manufacturer's data, 980300-004. Intel Corp., Santa Clara CA, undated.

50. Intel Corp. ISIS-II PL/M-86 Compiler Operator's Manual. Manufacturer's data, 980478-004. Intel Corp., Santa Clara CA, undated.

51. Intel Corp. MCS-86 Absolute Object File Formats. Manufacturer's data, 980821A. Intel Corp., Santa Clara CA, 1977.

52. Intel Corp. MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users. Manufacturer's data, 980639B. Intel Corp., Santa Clara CA, undated.

53. Intel Corp. Memory Components Handbook. Manufacturer's data, 210830-002. Intel Corp., Santa Clara CA, 1983.

54. Intel Corp. Microprocessor and Peripheral Handbook. Manufacturer's data, 210844-001. Intel Corp., Santa Clara CA, 1983.

55. Intel Corp. OEM Systems Handbook. Manufacturer's data, 210941-004. Intel Corp., Santa Clara CA, 1984.

56. Intel Corp. PL/M-80 Programming Manual. Manufacturer's data, 980268-002. Intel Corp., Santa Clara CA, undated.

57. Intel Corp. PL/M-86 Programming Manual. Manufacturer's data, 980466-003. Intel Corp., Santa Clara CA, undated.

58. Intel Corp. Prototyping with the 8089 I/O Processor. Manufacturer's data, AP-89. Intel Corp., Santa Clara CA, May 1980.

59. Intel Corp. Software Handbook. Manufacturer's data, 230766-001. Intel Corp., Santa Clara CA, 1984.

60. Intel Corp. Universal PROM Programmer User's Manual. Manufacturer's data, 9800319-01. Intel Corp., Santa Clara, 1978.

61. Intel Corp. 8039 Macro Assembler User's Guide. Manufacturer's data, 980938-02. Intel Corp., Santa Clara CA, 1980.

62. Lin, Shu, and Daniel J. Costello, Jr. Error Control Coding: Fundamentals and Applications. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1983.

63. Madnick, Stuart E. and John J. Donovan. Operating Systems. New York, New York: McGraw-Hill Book Co., 1974.

64. Matheson, William F. "Continued Development of a Universal Network Interface Device Using the INTEL 8086 and 8089 16-Bit Microprocessors." MS thesis, AFIT-GE-EE-83D-42. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.

65. Merwin, Richard E. Secure Operating System Technology Papers for the Seminar on the DoD Computer Security Initiative Program. American Federation of Information Processing Societies Conference Proceedings, National Computer Conference. Montvale NJ: American Federation of Information Processing Societies, Inc., June, July 1979.

66. Militar, Standard. File Transfer Protocol. MIL-STD-1780 (Draft), US Government Printing Office, Arlington, Virginia: 1983.

67. Military Standard. Internet Protocol. MIL-STD-1777, US Government Printing Office, Arlington, Virginia: 1983.

68. Military Standard. Simple Mail Transfer Protocol. MIL-STD-1781 (Draft), US Government Printing Office, Arlington, Virginia: 1983.

69. Military Standard. Transmission Control Protocol. MIL-STD-1778, US Government Printing Office, Arlington, Virginia: 1983.

70. Military Standard. TELNET Protocol. MIL-STD-1782 (Draft), US Government Printing Office, Arlington, Virginia: 1984.

71. Transport Layer Specification. Draft specification. Documentation and programs available on UNIX compatible magnetic tape. Washington DC, National Bureau of Standards, December 1983.

72. Hartrum, Thomas C. LSINET The AFIT Digital Engineering Laboratory (DEL) Network of LSI-11 and PDP-11 Computers. AFIT DELNET Documentation, Version 3.1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1984.

73. Palmer, Donald E. "Design of a Prototype Universal Network Interface Device Using INTEL 8086 and 8089 16-Bit Microprocessors." MS thesis, AFIT-GCS-EE-82D-52. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.

74. Papp, Charles E. "Prototype DELNET Using the Universal Network Interface Device." MS thesis, AFIT-GE-EE-81D-46. School of Engineering, Air Force Institute of Technology (AU , Wright-Patterson AFB OH, December 1981.

75. Phister, Paul W. Jr. "Protocol Standard and Implementation Within the Digital Engineering Laboratory Computer Network (DELNET) Using the Universal Network Interface Device (UNID)." MS thesis, AFIT-GE-EE-83D-58. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.

76. Pickens, R. Andrew. "Wideband Transmission Media II: Satellite Communications," Computer Communications, Volume 1, Principles, edited by Wushow Chou. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1983.

77. Pingry, Julie. "Local Area Networking Becomes A Standard Feature," Digital Design, 14: 70-83 (March 1984).

78. Ravenscroft, D. "Electrical Engineering Digital Design Laboratory Communications Network." MS thesis, AFIT-GCS-EE-78-16. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1978 (AD A064729).

79. Rivest, Ronald L., Adi Shamir and Len Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Massachusetts Institute of Technology publication MIT/LCS/TM-82. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, April 1977.

80. Rubin, Izhak and Luis F. M. De Moraes. "Message Delay Analysis for Polling and Token Multiple-Access Schemes for Local Communication Networks," IEEE Journal on Selected Areas in Communications, SAC-1: 935-946 (November 1983).

81. Sauer, Charles H. and K. Mani Chandy. Computer Systems Performance Modeling. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.

82. Sauer, Charles H. and Edward A. MacNair. Simulation of Computer Communication Systems. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1983.

83. Sinkov, Abraham. Elementary Cryptanalysis. Washington DC: Mathematical Association of America, 1966.

84. Sluzevich, Sam C. "Preliminary Design of a Universal Network Interface Device." MS thesis, AFIT-GE-EE-78-41. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1978 (AD A064059).

85. Spear, Mark C. "Hardware Design and Implementation of the Universal Network Interface Device (UNID)." MS thesis, AFIT-GE-EE-84M-XX. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1984.

86. Steinmetz, Jay S. "A Secure Computer Network." MS thesis, AFIT-GCS-EE-82D-34. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1982.

87. Tannenbaum, Andrew S. Computer Networks. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.

88. Witt, Michael. "An Introduction to Layered Protocols." Byte: 385-398, September 1983.

89. Zilog, Inc. PLZ User Guide. Manufacturer's data, 03-3096-01. Zilog Inc., Cupertino CA, July 1979.

Appendix A

UNID II Data Flow Diagrams

This appendix contains the Data Flow Diagrams (DFD) for the UNID II which were developed in a previous thesis effort (30:26-34). These DFDs show the UNID II message processing functions and the internal flow of messages between the local and network I/O hardware ports. The DFDs are still current and are shown in this thesis for completeness. The DFDs are presented in the following order:

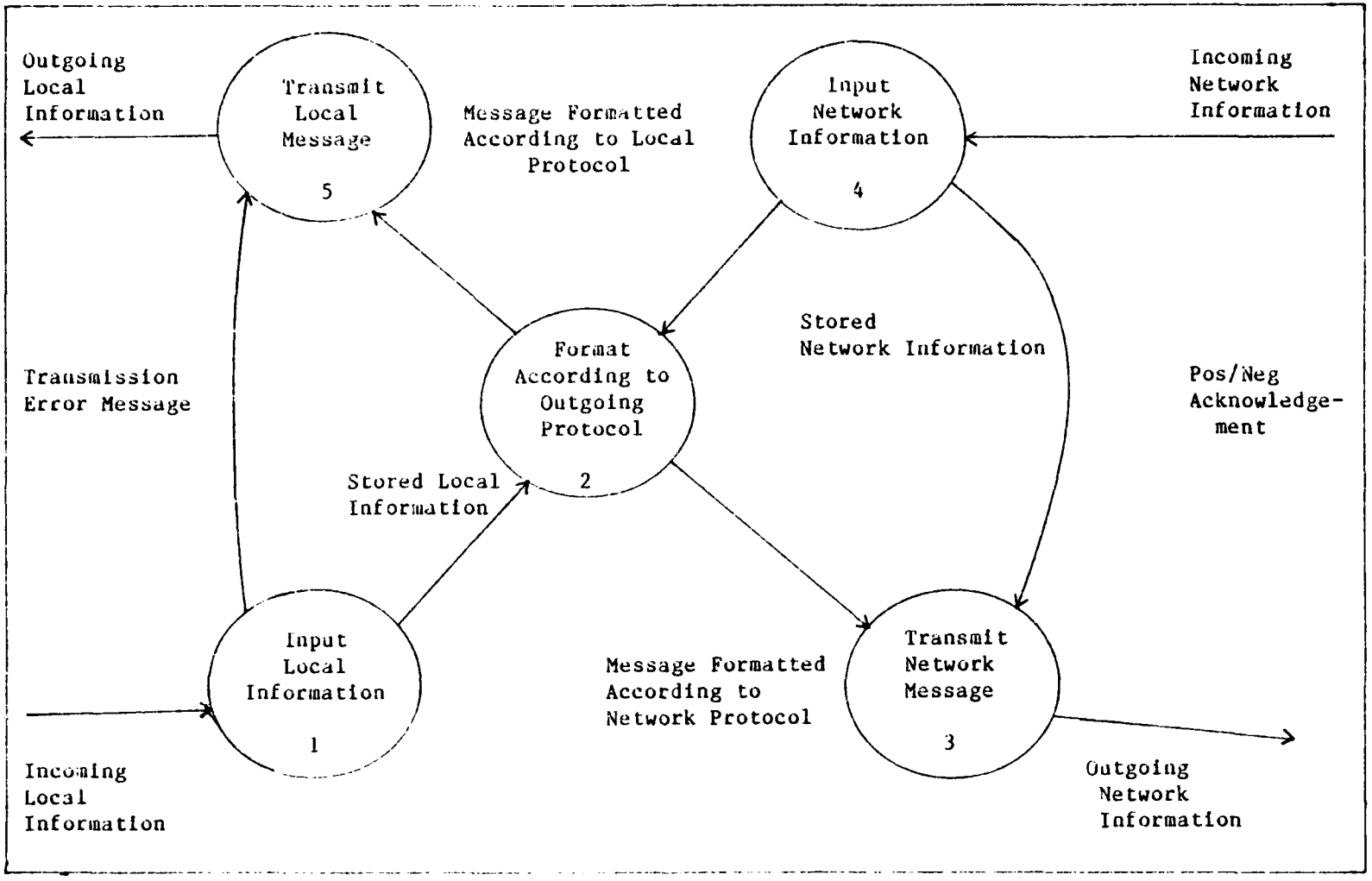


Figure A-1. UNID II Overview (30)

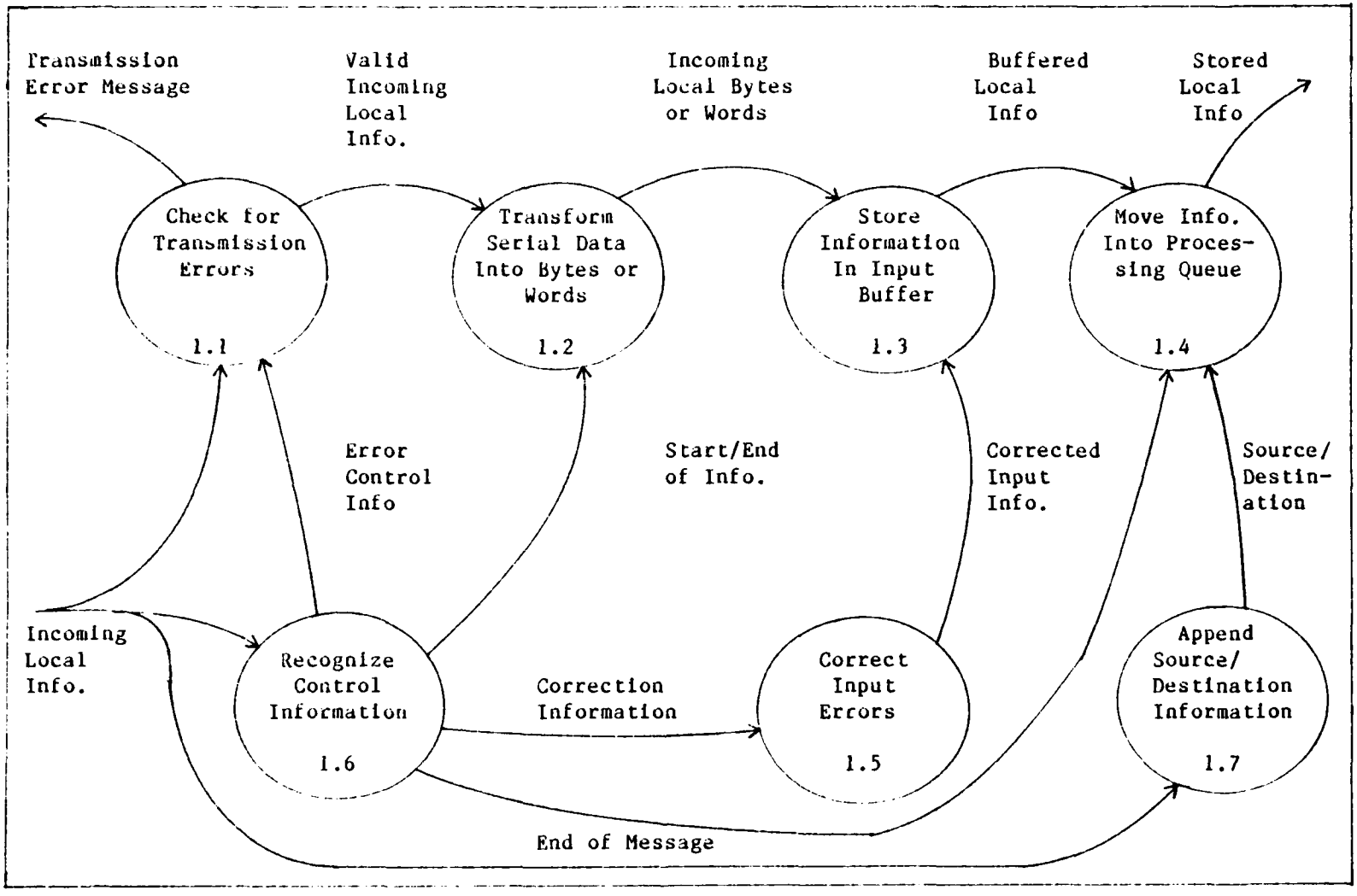


Figure A-2. Input Local Information (30)

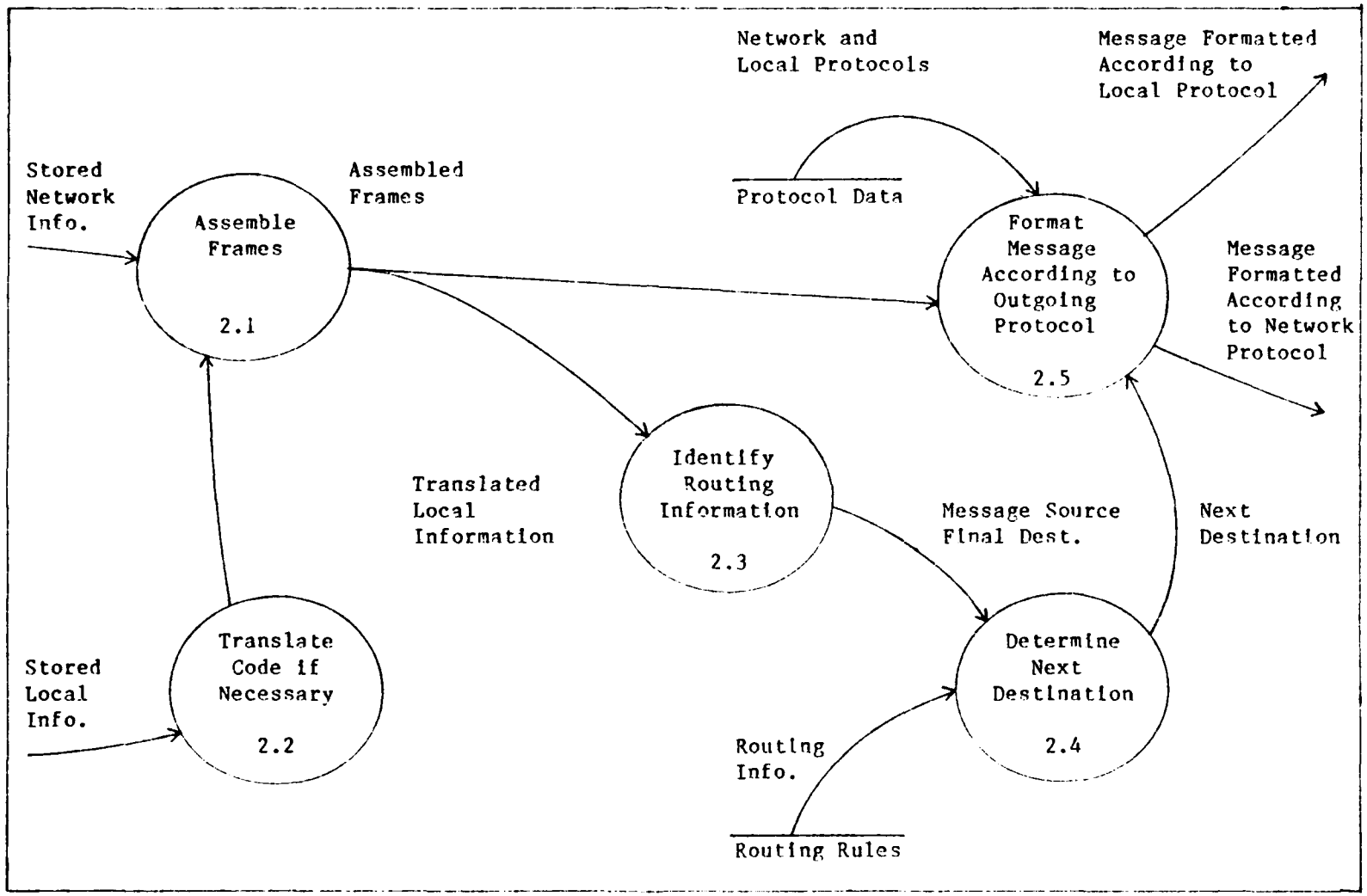


Figure A-3. Format According to Outgoing Protocol (30)

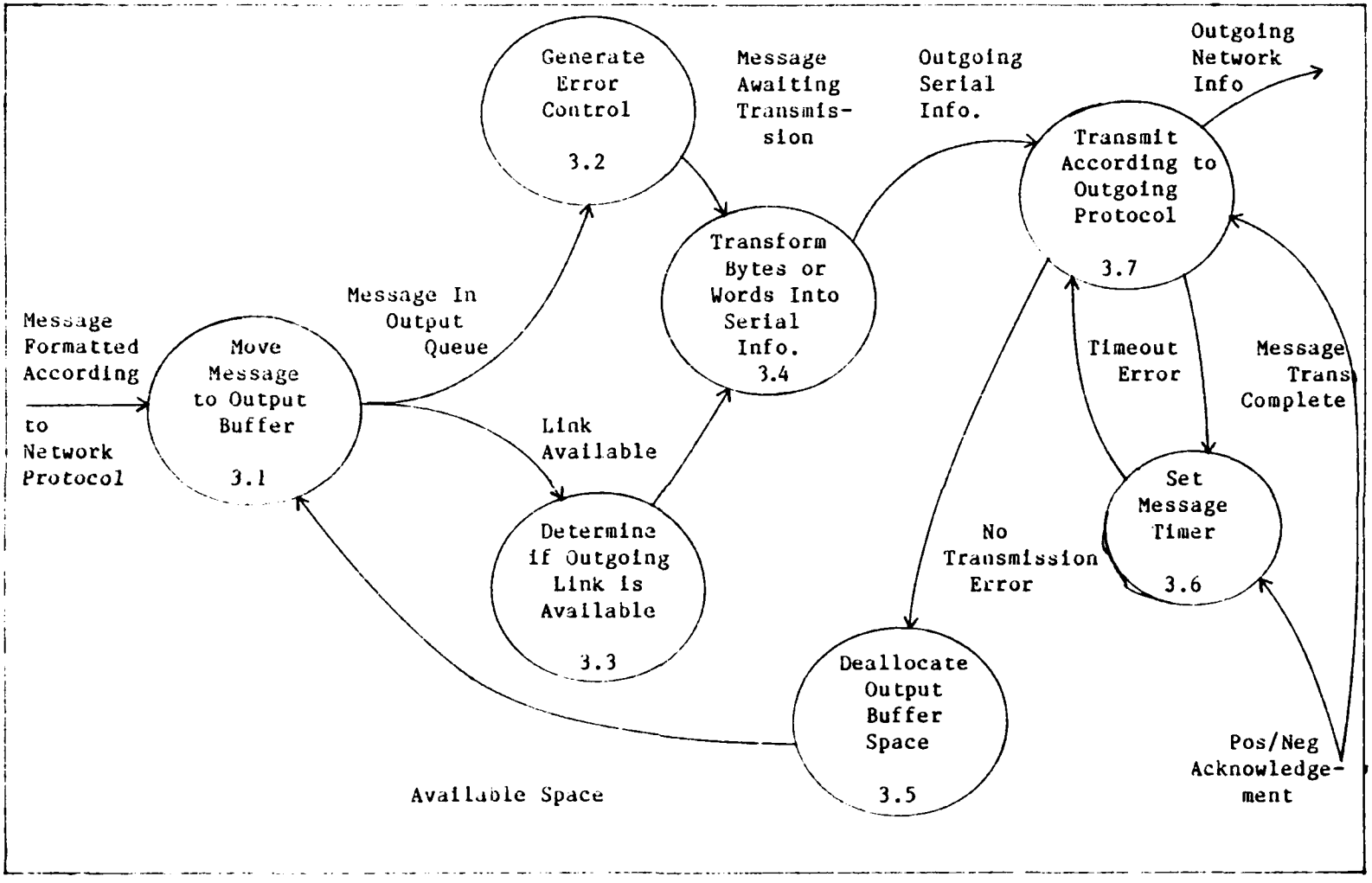


Figure A-4. Transmit Network Message (30)

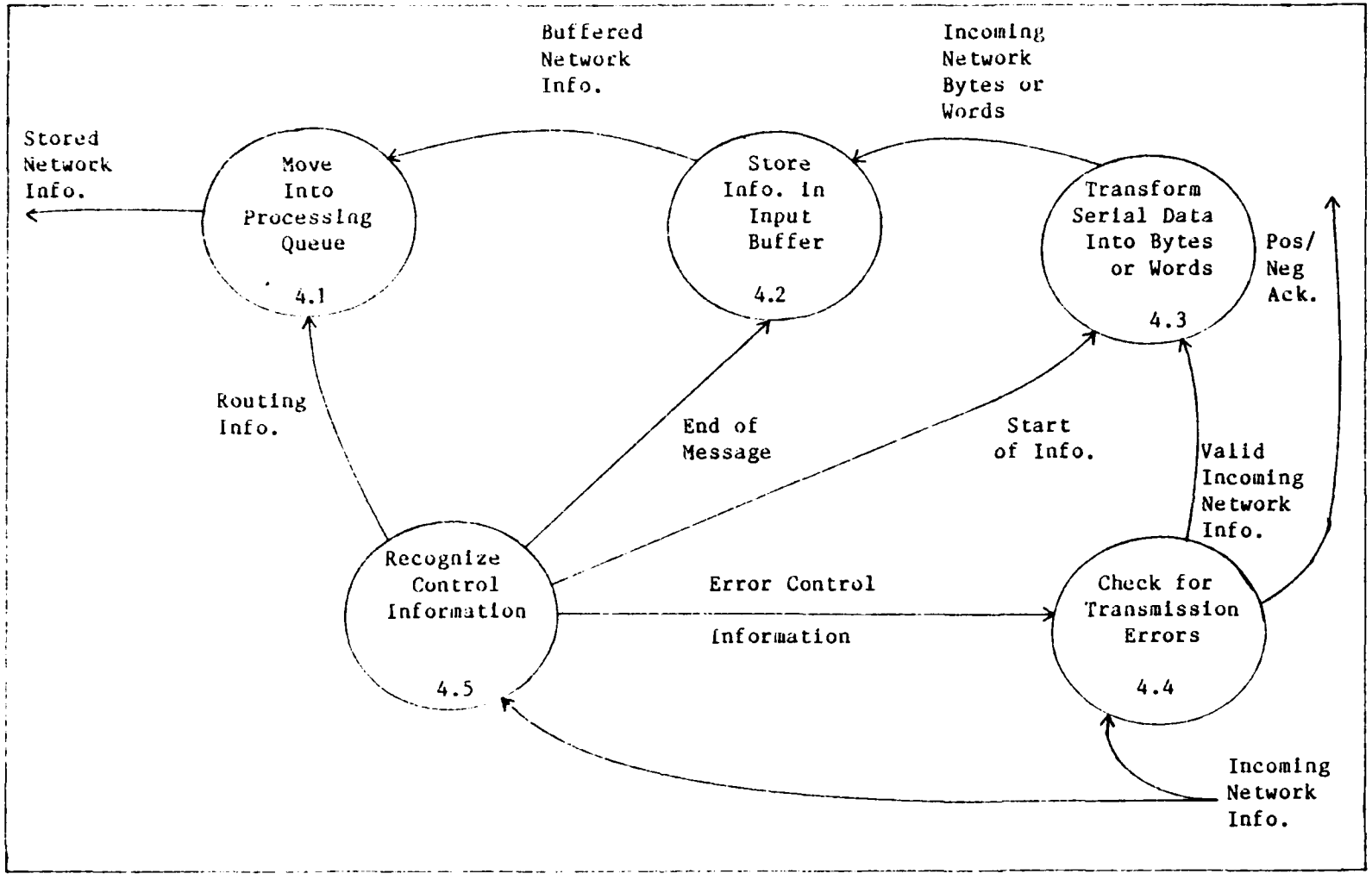


Figure A-5. Input Network Information (30)

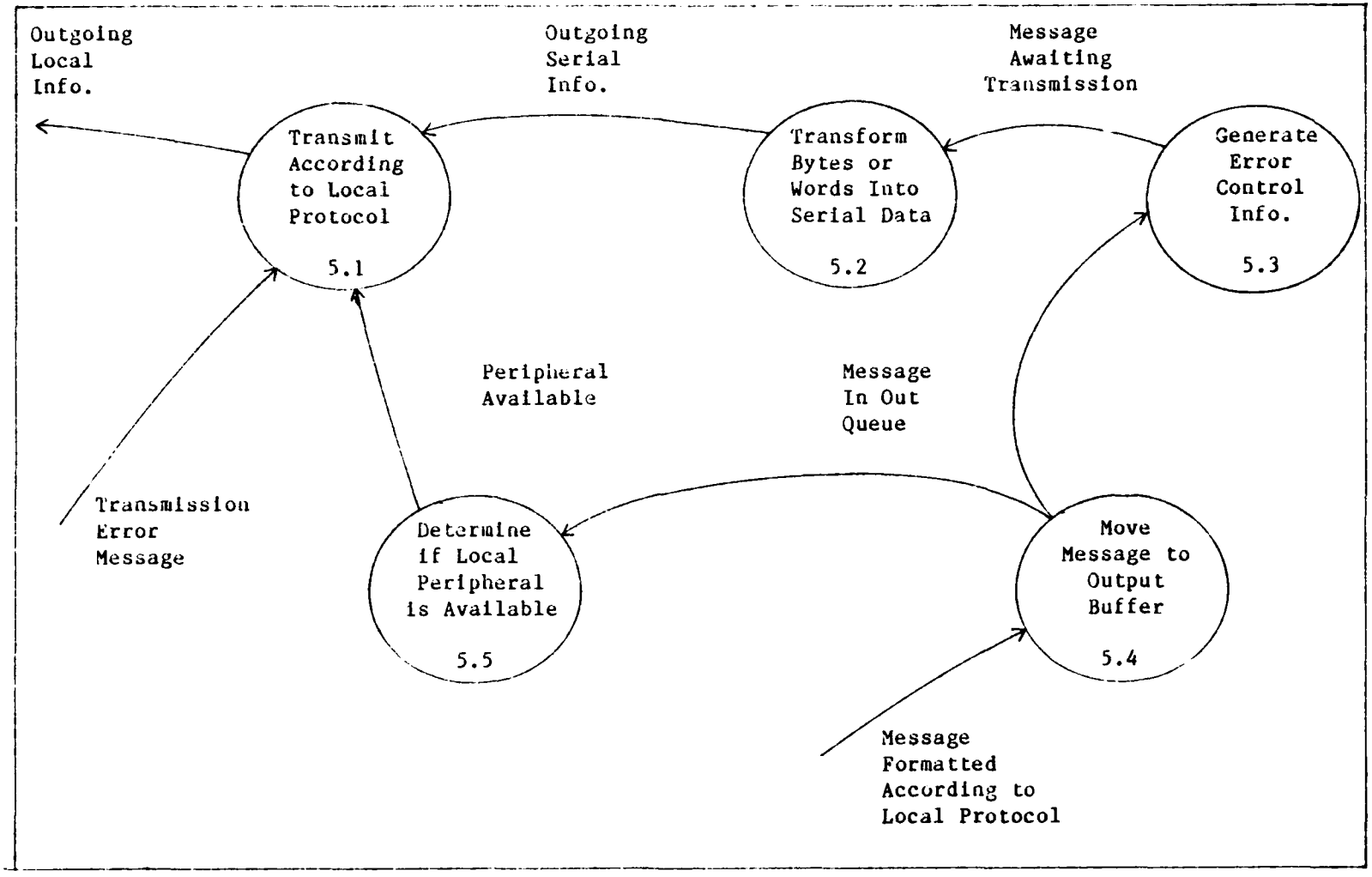


Figure A-6. Transmit Local Message (30)

Appendix B

RS-232C and RS-422 Signals

The network layer (SBC 544 or similar hardware interface on the local host side of the UNID II) will use the RS-232C standard to interface with the host computers or other DELNET networks (75:Chap 1, Chap 2), such as the NETOS (72). The data link layer (SBC 88/45 or similar hardware interface on the network side of the UNID II) will use the RS-422 standard to interface one UNID with another UNID (75:Chap 1, Chap 2).

Each standard (18, 19) has a large number of signals specified not all of which are used by the UNID IIs (75: Appen B). The following figures indicate the signals that are implemented in the UNID and DELNET. Signal direction into or out of the UNID II is shown for the RS-422 interface. Signal direction into or out of the UNID II is not shown for the RS-232C interface as the SBC 544 board is jumper configureable for either a DTE or a DCE connection. The default configuration for the RS-232C ports on the SBC 544 board is set to a DCE to allow most host computers and terminals to directly connect to the UNID II without the use of a null modem.

DB-25 Pin Number	Signal Nomenclature	Implemented in DELNET
1	Frame Ground	X
2	Transmit Data	X
3	Receive Data	X
4	Request to Send	X
5	Clear to Send	X
6	Data Set Ready	X
7	Signal Ground	X
8	Receive Line Signal Detect	(X)
9	Unassigned	
10	Unassigned	
11	Unassigned	
12	Secondary Receive Signal Detect	
13	Secondary Clear to Send	
14	Secondary Transmit Data	
15	Transmit Signal Element Timing	
16	Secondary Receive Data	
17	Receive Signal Element Timing	
18	Unassigned	
19	Secondary Request to Send	
20	Data Terminal Ready	X
21	Signal Quality Indicator	
22	Ring Detector	(X)
23	Data Signal Rate Select (DTE)	
24	Data Signal Rate Select (DCE)	
25	Unassigned	

NOTE: Pins 3 and 22, marked (X), are not required when the UNID is hard wired to a host computer or terminal. These pins would normally be used when a particular port is connected to a modem.

Figure B-1. RS-232C Pin Assignments

The local side connected to host computers or terminals will use the RS-232C signals shown in Figure B-1. It is assumed that the RS-232C interface uses a standard DB-25 connector.

DB-37 Pin Number	Signal Nomenclature	Direction In	Direction Out	Implemented in DELNET
1	Shield			X
2	Signal Rate Indicator A			
3	Spare B			
4	Send Data A		X	X
5	Send Timing A		X	X
6	Receive Data A	X		X
7	Request to Send A		X	X
8	Receive Timing A	X		X
9	Clear to Send A	X		X
10	Local Loopback A			
11	Data Mode A			
12	Terminal Ready A		X	X
13	Receiver Ready A	X		X
14	Remote Loopback			
15	Incoming Call	X		
16	Select Frequency Signalling Rate Indicator			
17	Terminal Timing		X	
18	Test Mode A			
19	Signal Ground			X
20	Receive Common			
21	Spare A			
22	Send Data B		X	X
23	Send Timing B		X	X
24	Receive Data B	X		X
25	Request to Send B		X	X
26	Receive Timing B	X		X
27	Clear to Send B	X		X
28	Terminal in Service A			
29	Data Mode B			
30	Terminal Ready B			X
31	Receiver Ready B			X
32	Select Standby A			
33	Signal Quality		X	
34	New Signal			
35	Terminal Timing		X	
36	Standby Indicator			
37	Send Common			

NOTE: The -A and -B suffixes on the signal nomenclature refer to the non-inverted and inverted outputs/inputs of the RS-422 signals as the signals use a balanced signalling method and a reversed connection will invert the desired signal.

Figure B-2. RS-422 Pin Assignments

The UNIDs are connected using the RS-422 standard on the data link layer (subnetwork) side of the UNID. Figure B-2 shows the pin assignments and indicates which signals are currently implemented in the DELNET. In this figure, the direction into and out of the UNID is also shown to clarify the use of a null modem required on one of the two high speed network ports. It is assumed that the RS-422 interface uses a standard DB-37 connector.

Implementation Corrections in Previous Thesis

Introduction

This appendix explains implementation errors, and their corrections, from previous work (29, 64, 75). The hardware errors were detected during the early analysis of the previous designs (64). The hardware errors refer to the revised network (data link layer) board designed by (64). While this design was not used in this effort, the errors are noted for future reference. The software errors were discovered during the early analysis of the previous designs and during the network and data link layer software simulations. All the detected errors have been corrected.

Software Corrections

The toggling of the sequence bit in the transmitted frame for the one bit sliding window protocol was forgotten. The sequence bit is never toggled in the I frames, even though the ROUTE$OUT procedure does toggle the sequence bit on a true ACK. Since ROUTE$IN checks the input sequence bit against the transmitted sequence bit to determine a true acknowledgement, a true acknowledgement will only occur on the first transmitted frame. The software will then hang up resending the second frame ad infinitum. The corrective action is to include the sequence bit in the BUILDIFRAME procedure.

A counter and error recovery for the maximum number of retransmissions in the ROUTE$OUT procedure is needed. There currently is none. The result is that if a frame is never acknowledged and the timer times out, the software will hang up sending the frame forever! The corrective action is to include a retransmission counter which will

perform a specified action if the maximum number of retransmissions is exceeded.

The design for interpreting the incoming frame in the NTO1TB and NTO2TB tables is incorrect. The current design first detects if the incoming frame is to continue around the network (NN) or is destined for the local UNID (NL). If the frame is NN, then an S frame is sent to the sender of the frame and the received frame is then sent around the loop. This situation is correct for an I frame, but causes havoc with an S frame. In effect, S frames are generated as acknowledgements for other S frames and consequently, S frames will be generated at a geometric rate and will choke the network. The NL logic is satisfactory. The solution is to interpret the incoming frame to determine if it is an I or S frame, then determine it's destination. This solution will prevent S frame flooding and route the frames correctly.

A problem exists in the ROUTE$OUT procedure when ACKNOWLEDGE$A(B) is true. The frame to send is correctly discarded, however, ACKNOWLEDGE$A(B) is never reset false, the timeout flag and counter are not reset, and the number of retransmissions counter, assumed to be implemented, is not initialized. The ACKNOWLEDGE$A(B) reset to false was placed in the incorrect place. This will only allow sending the first frame whereupon the software will not send any other packets. The corrective action is to include the proper resetting of the flags and counters when the ACKNOWLEDGE$A(B) is true.

An error with the implementation logic of the one bit sliding window timeout and the acknowledge exists. Both timeout and acknowledge need to be considered together (87:43), not separately as implemented (75). The software design and implementation can be derived from the

truth table in Figure C-1. The corrective action is to implement the software according to the truth table.

ACK	TIMEOUT	FUNCTION
0	0	Increment TIMOUT counter
0	1	(re)send a frame
1	0	service transmit table, reset flags and counters
1	1	same as the (1, 0) case

Figure C-1. Acknowledge and Timeout Truth Table

The INPUTSEQBIT and THISSEQBIT require the suffix `$A` and `$B` to separate the sequencing of the I and S frames for the A and B loops , otherwise the software becomes confused when both the A and B directions are transmitting and receiving frames. The corrective action is to implement the INPUTSEQBIT and THISSEQBIT for both the $A and $B sections of software.

The original software design does not insure that one and only one I frame is in the OUTFRAME$CHA(B)$TB table at a given instant of time. The one bit sliding window protocol requires that the current I frame be retained until a positive acknowledgement is received. If two or more I frames are in the transmit buffer at the same time, the transmit software will send them both and subsequently confuse the sequence bits. This results in the software continually retransmitting the frames until the maximum retransmit counter is exceeded. The corrective action is to generate a procedure to determine if an I frame is in the transmit buffer and inform the calling procedure accordingly. The calling

procedure should then load an I frame only when there is no I frame in the transmit buffer.

An error exists with the implementation of the bit masking used for the detection of the acknowledge frames and the sequence bits. The bit masking was implemented in reverse order, that is the eighth bit is masked for detection of an S frame where the first bit should be masked. Similarly, the sixth bit is masked for detection of the sequence bit where the third bit should be masked. The source of the problem is interpretation between the left to right ordering of the bits in the source documentation and the left to right implementation in the software. The source documentation shows the least significant bit on the left and the most significant bit on the right, however the bit mask coding is just reversed with the most significant bit on the left and the least significant bit on the right. This reversal did not affect the original implementation because the software was the same at both ends of the UNID connection and each end saw the correct information where it should have. Consequently, this error was not previously detected. The corrective action is to implement the bit masking correctly.

The BUILDIPACKET procedure in the original network layer software contained a case statement that performed the same assignment statements regardless of the case selector (64, 75). While this is not an error to the software as it then existed, it did use extra memory. The procedure was rewritten using pointers and semaphores as explained in Chapter Four.

Hardware Corrections

A design error exists in network board schematic. Data lines D8 - D15 and D0 - D7 in the memory array need to be controlled by A0 and BHE* (generated by the 8089 processor), otherwise the memory will not be accessed correctly for byte read/write operations. The corrective action is to redesign the memory data control similar to the SBC 86/12A logic to properly gate the memory data.

A design or typographical error exists with the address decoding lines for the 8089 local I/O space. The address lines A14 and A15 are reversed. The corrective action is to implement the addressing correctly and annotate the schematic diagram.

The RS 232/422 DTE/DCE strapping options shown on the schematic are in the incorrect locations. They need to be between the connectors and drivers, not between the USART and the drivers, otherwise the drivers are connected output to output and input to input with the USART. The corrective action is to redesign the strapping options in the correct location and annotate the schematic diagram.

Appendix D

DELNET/UNID Header Information

This appendix expands upon the TCP/IP datagram, packet, and frame header information presented briefly in Chapter One (75:Appen C). Each byte in a complete datagram, packet, and frame is shown with the appropriate bit information within each byte. The name of each byte position, along with the array index number, is given for each byte. The subscripts H and L refer to most significant byte and least significant byte, respectively. Similarly, the subscripts 3, 2, and 1 indicate the most significant byte, the next significant byte and the least significant byte, respectively. The contents of each byte conform to the standards established in (28, 67, 69, 75:Appen C). Entries that contain letters refer to specific bits that must be initialized or set according to how and when they are used. For example, the packet source address will be a constant that depends upon the particular UNID and port number to which a host is connected. The Type of Service byte on page D-2 depends upon the precedence, delay, throughput and routing required by the transport and higher level protocols. Those bits and bytes that contain letters are variables that are data and user dependent. Those bytes that are empty are not yet used by the UNID II software and are currently filled with zeros. The particular mapping shown was used for the testing of the UNID II in the LSI-11 NETOS test. Each byte is further explained in (28, 67, 69, 75:Appen C).

Name	Index: Frame	Index: Packet	Index: Datagram	Bits: MSB … LSB
				0 0 0 0 | 0 0 0 0
Unid Dest/Source	0			d d d d | s s s s
Control s = 0,1 => sequence c = 1 => control	1			0 0 0 u | 0 0 s c
Destination Addr u = unid # c = channel #	2	0		u u u u | c c c c
Source Address u = unid # c = channel #	3	1		u u u u | c c c c
Sequence #	4	2		n n n n | n n n n
Spare(1)	5	3		0 0 0 0 | 0 0 0 0
Spare(0)	6	4		0 0 0 0 | 0 0 0 0
IP Header	7-38	5-36	0-31	
Vers #/IHLength	7	5	0	0 1 0 0 | 1 0 0 0
Type of Service p = precedence p = 000 => routine datagram p = 110 => Internet Control	8	6	1	p p p d | t r 0 0
Total Length(H)	9	7	2	0 0 0 0 | 0 0 0 0
Total Length(L)	10	8	3	1 0 0 0 | 0 0 0 0
User Ident(H)	11	9	4	u u u u | u u u u
User Ident(L)	12	10	5	u u u u | u u u u
Flags/Frag Off(H)	13	11	6	0 1 0 | 0 0 0 0 0
Fragment Offset(L)	14	12	7	0 0 0 0 | 0 0 0 0
Time to Live (60)	15	13	8	0 0 1 1 | 1 1 0 0
Protocol	16	14	9	0 0 0 0 | 0 1 1 0
Header Checksum(H)	17	15	10	0 0 0 0 | 0 0 0 0

Name	Index Frame	Packet	Datagram	Bits MSB LSB
Header Checksum(L)	18	16	11	0 0 0 0 | 0 0 0 0
Source Address:				
Control/Country	19	17	12	0 0 0 0 | 1 0 0 1
Network Code(UNID #) Host Code(H)	20	18	13	0 0 1 0 | h h h h
Host Code(L)/ Port Code(2)	21	19	14	h h h h | p p p p
Port Code(1)/ Port Code(0)	22	20	15	p p p p | p p p p
Destination Address:				
Control/Country	23	21	16	0 0 0 0 | 1 0 0 1
Network Code(UNID #) Host Code(H)	24	22	17	n n n n | h h h h
Host Code(L)/ Port Code(2)	25	23	18	h h h h | p p p p
Port Code(1)/ Port Code(0)	26	24	19	p p p p | p p p p
Security(H)	27	25	20	1 0 0 0 | 0 0 1 0
Security(L)	28	26	21	0 0 0 0 | 1 0 1 1
S Field(H)	29	27	22	
S Field(L)	30	28	23	
C Field(H)	31	29	24	
C Field(L)	32	30	25	
H Field(H)	33	31	26	
H Field(L)	34	32	27	
TCC Field(2)	35	33	28	
TCC Field(1)	36	34	29	
TCC Field(0)	37	35	30	

Name	Index Frame	Index Packet	Index Datagram	Bits MSB LSB
LHF Padding	38	36	31	0 0 0 0 | 0 0 0 0
TCP Header	39-62	37-60	32-55	
Source Port(H)	39	37	32	
Source Port(L)	40	38	33	
Destin Port(H)	41	39	34	
Destin Port(L)	42	40	35	
Sequence #(3)	43	41	36	
Sequence #(2)	44	42	37	
Sequence #(1)	45	43	38	
Sequence #(0)	46	44	39	
Acknowledge #(3)	47	45	40	
Acknowledge #(2)	48	46	41	
Acknowledge #(1)	49	47	42	
Acknowledge #(0)	50	48	43	
Data Offset/Resv	51	49	44	0 1 1 0 | 0 0 0 0
Reserved/Control	52	50	45	0 0 u a | p r s f
Window(H)	53	51	46	
Window(L)	54	52	47	
Checksum(H)	55	53	48	
Checksum(L)	56	54	49	
Urgent Ptr(H)	57	55	50	
Urgent Ptr(L)	58	56	51	
Option	59	57	52	
Padding(2)	60	58	53	0 0 0 0 | 0 0 0 0

.pa

Name	Index			Bits							
	Frame	Packet	Datagram	MSB							LSB
Padding(1)	61	59	54	0	0	0	0	0	0	0	0
Padding(0)	62	60	55	0	0	0	0	0	0	0	0
User Data	63-134	61-132	56-127	x	x	x	x	x	x	x	x

Figure D-1. DELNET/UNID Detailed Header Information

Appendix E

UNID Semaphores and Protected Regions

This appendix explains in detail the use of semaphores as implemented for the exchange of information between the SBC 544 and SBC 88/45 boards.

The use of semaphores is required to protect a critical region (13:78) of program execution from being disturbed by other concurrent processes in a multiprocess or multiprocessor environment. As was earlier noted, the UNID I and II are multiprocessor and multiprocess systems. Data in the form of packets are exchanged between the two processors. The process on the SBC 544 board which alerts the SBC 88/45 board that a packet is ready for the SBC 88/45 process must ensure that the SBC 88/45 process is not manipulating the last set of data left by the SBC 544 process, otherwise the SBC 544 process will, in all likelihood, write over the old data with the new data and cause inadvertent destruction of desired data (13:77).

Equally as important, the SBC 88/45 process must ensure that the SBC 544 process is not manipulating data in the critical region of memory when the SBC 88/45 process wants access to the critical region. The method to ensure a writeover (race) condition does not exist often involves the use of P and V type semaphore operators (13:89). The P and V operators are often implemented with low level hardware operations, such as TestAndSet which is called LOCK for the Intel processors, that will interrogate and set a flag to ensure that only one process is manipulating data in a critical region of memory at one time. Unfortunately, as mentioned in Chapter Three of this thesis, the SBC 544

board does not have the TestAndSet (LOCK) capability even though the SBC 88/45 board does. The SBC 544 board also does not have the mechanisms to allow the SBC 88/45 board to use its LOCK operator. Therefore another method was devised which allows both boards to share certain portions of the common system memory while allowing only one processor and process to access that shared, critical region at one time, thus preventing inadvertent destruction of data (13:77). This method uses a variable with only two states, Ready and Done, as the semaphore. The SBC 544 process will check the variable for the Done state, and if Done, will update the packet pointer and set the semaphore to the Ready state. The SBC 88/45 process checks for the Ready state, and if Ready, will move the packet to another buffer for further processing and set the semaphore to the Done state. The SBC 544 process is allowed only to check the semaphore for Done and set the semaphore to Ready. The SBC 88/45 process is allowed only to check the semaphore for Ready and set the semaphore to Done. By implementing the processes in this manner, the processes will stay in synchronization and not manipulate data until the data is ready to be manipulated. Each process is not allowed to wait until the other process is completed; it will continue performing other tasks and will, at some later time, reenter the semaphore testing routine. This mechanism is illustrated in Figure E-1 with the aid of pseudocode.

```
If DatagramAvailable then          (Process executed by SBC 544)
     If LSem = Done then
          do;
          LPointer = .LocalReceive(NexttoSend)
          LSem = Ready
          service(.LocalReceive(NexttoSend))
          end;


If LSem = Ready then               (Process executed by SBC 88/45)
     do;
     move(LPointer, .NetworkTransmit(NextEmpty), PacketSize)
     LSem = Done
     load(.NetworkTransmit(NextEmpty))
     end;
```

Figure E-1. Pseudocode for SBC 544 to SBC 88/45 Packet Movement

The first section of pseudocode corresponds to the process on the SBC 544 board and the second section of pseudocode corresponds to the concurrent process on the SBC 88/45 board. DatagramAvailable is an indication that a packet is available to move to the network board, LPointer is a pointer to the available packet, LSem is the semaphore, .LocalReceive(NexttoSend) is the pointer to the available packet in the SBC 544 memory, .NetworkTransmit(NextEmpty) is the pointer to the next available entry for a packet in the SBC 88/45 memory. LPointer and LSem, as well as the packet, are stored in the shared system memory. The routines `service' and `load' adjust the pointers within the tables given as arguments and `move' moves a specified amount of data from one location to a second location. The reader should recall that the tables LocalReceive, LocalTransmit, NetworkReceive and NetworkTransmit used in the UNID software are circular FIFO queues whose first-in pointer is NextEmpty and first-out pointer is NexttoSend.

The above high level code is divisible into smaller sets of

indivisible assembly language code, each of which can be interrupted by other processes on the respective SBC 544 or SBC 88/45 board. The sections of code that can interrupt the above processes, however, do not manipulate any data used by the above sections of code, and will therefore not disturb the critical sections except for inducing a non critical time delay. In addition, the remaining processes on each board never manipulate the semaphore. The LocalReceive buffer is used by other SBC 544 processes. These are the host receive interrupt routine, where the table pointer NextEmpty is manipulated, and the send a datagram from local host to local host routine where the table pointer NexttoSend is manipulated. The latter routine, while manipulating the NexttoSend pointer, will only do so when the particular datagram is for local to local host movement. The particular routine of which the above code is a part interrogates the IP header of the first datagram pointed to by NexttoSend. If this datagram is for local host to local host movement, the datagram is moved and the NexttoSend pointer updated. However, if the datagram is for local host to network movement, the first section of pseudocode above is called. The pointer NexttoSend will only be updated if the SBC 88/45 is ready for another packet, otherwise the pointer is left untouched and the datagram in question is still at the top of the LocalReceive table waiting to be moved. So while the NexttoSend pointer can be manipulated by another process, it is done so only if the datagram in question at the top of the LocalReceive table is going back to a local host and not to the network. Therefore the NexttoSend pointer will be updated for a network destined packet only if the datagram in question at the top of the LocalReceive table is going to the network when the first section of the pseudocode

is entered.

It should be noted that while both processors have the ability to access the shared system memory at the same time, and will probably attempt to do so, they cannot, in fact, read (or write) to the same shared location at exactly the same instant. This `read at the same instant´ phenomena is prevented by the hardware design of both boards. If the SBC 544 processor is in the middle of a fetch from shared memory, the hardware design locks out access to the memory to other processors with access to that shared memory. Therefore, the SBC 88/45 board cannot access the desired location until the SBC 544 board has completed its current instruction, whereupon, the SBC 88/45 may then access the shared memory. The same holds true for an access of shared memory by the SBC 88/45 board. When the SBC 88/45 processor accesses shared memory, other processors are locked out from accessing the same shared memory until the SBC 88/45 has completed its current instruction. Therefore, if the SBC 88/45 is executing the instructions to set `LSem = Done´, and the SBC 544 is fetching the `LSem´ location to interrogate for `Done´, there will not be a simultaneous access of the location `LSem´ as explained above. Therefore the SBC 544 interrogation of `LSem´ will find its value either `Done´ or `Ready´ as expected and execute the critical section accordingly.

The process communication from the SBC 88/45 board to the SBC 544 board is identical to that for the SBC 544 to SBC 88/45 board explained above. The variable names are different so as to keep the two processes and functions separated. The two processes function the same and the discussion above applies to the SBC 88/45 to SBC 544 board communication. The pseudocode for the SBC 88/45 to SBC 544 board

communication is shown in Figure E-2.

```
If PacketAvallable then          (Process executed by SBC 88/45)
     If NSem = Done then
          do;
          NPointer = .NetworkReceive(NexttoSend)
          NSem = Ready
          service(.NetworkReceive(NexttoSend))
          end;


If NSem = Ready then             (Process executed by SBC 544)
     do;
     move(NPointer, .NetworkReceive(NexttoSend), DatagramSize)
     NSem = Done
     load(.LocalTransmit(NextEmpty))
     end;
```

Figure E-2. Pseudocode for SBC 88/45 to SBC 544 Packet Movement

Appendix F

Transmit Request/Transmit Acknowledge Handshake

This appendix explains the details of the transmit request/transmit acknowledge mechanism implemented in the SBC 544 local host software.

The transmit request/transmit acknowledge mechanism is used to synchronize the UNID II network layer software with a comparable process on a local host. The mechanism allows the orderly transmission of datagrams between the host and the UNID II. This allows the UNID II to reliably send and receive datagrams from a host that is slower than the speed of the UNID II, whether the host polls the receive port or has a slow interrupt response, or a host that does not have an interrupt driven receive port from the UNID II. This mechanism, or something similar such as the DTR-DSR or RTS-CTS hardware handshake, must be implemented to allow the orderly transmission of datagrams between the UNID II and its connected hosts. The particular implementation explained below was chosen to accommodate the NETOS (LSI-11) network in the DELNET, which uses a software transmit request/transmit acknowledge scheme. The NETOS uses does not use a hardware handshake such as the RTS-CTS because the signals are not implemented on that system. Other mechanisms, such as XON-XOFF, may be relatively easily implemented in the UNID II software to accommodate similar the mechanisms in other networks.

The particular mechanism used by the NETOS can be described as follows (72). When a node in the NETOS desires to send a packet to another node, it first sends a transmit request (TR) to the desired node. The receiving node then, when it recognizes a TR was sent to it,

will send a transmit acknowledge (TA) back to the sender when it is ready to receive a packet. The sending node, when it recognizes the TA from the receiving node, sends the datagram to the receiving node. There is no final acknowledge sent by the receiver back to the sender to acknowledge the reception of the datagram. The TR/TA mechanism effectively reduces a normally full duplex channel to a half duplex channel. The process can be diagrammed as shown in Figure F-1.

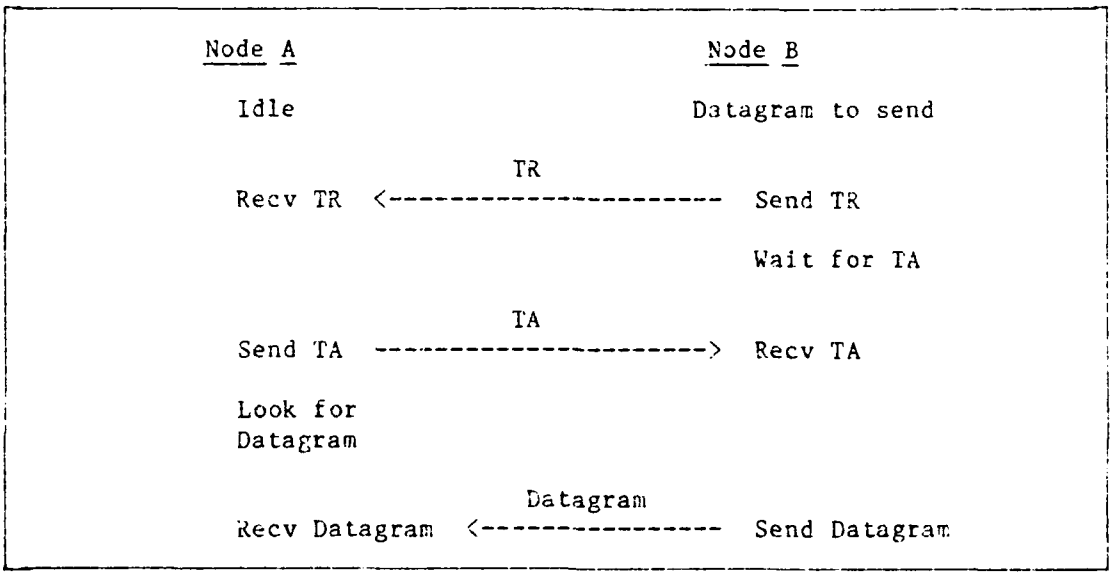


Figure F-1. NETOS Transmit Request/Transmit Acknowledge Handshake.

The mechanism is implemented in the UNID II software with four boolean flags for each host port. Four flags are required since both the NETOS node and the UNID will send and receive datagrams using the TR/TA mechanism, providing a full handshake for each direction. The UNID must know which state it is in so that it can communicate correctly with the NETOS node. The four flags are Transmit transmit request (TXTR), Receive transmit acknowledge (RXTA), Receive transmit request (RXTR), and Transmit transmit acknowledge (TXTA). Each flag has the

value TRUE or FALSE. The initial state is all four flags FALSE. Of the 16 possible states, the five allowed states are shown in Figure F-2. A `0' represents FALSE and a `1' represents TRUE.

TXTA	RXTA	RXTR	TXTA	
0	0	0	0	Initial state
1	0	0	0	Datagram to send
1	1	0	0	OK to send datagram
1	1	0	0	Send the datagram
0	0	0	0	Reset the flags after sending datagram
0	0	1	0	Datagram receive request
0	0	1	1	OK to receive datagram
0	0	1	1	Receive datagram
0	0	0	0	Reset flags after receiving datagram

Figure F-2. UNID TR/TA Allowable States.

Note that the mechanism starts in the all zero, or FALSE, state with no datagrams to send or receive and returns to the all zero state at the completion of sending or receiving a datagram. Also, only one process of sending a datagram or receiving a datagram is allowed at one time. While this is a requirement for the NETOS, and possibly other networks, the UNID software is totally interrupt driven and can send and receive a datagram simultaneously without this handshake mechanism. The UNID II local software on the 544 board has been designed through the use of a boolean flag labeled TRTA so that the TR/TA half duplex handshake may be used or not used on any given host port of the 544

board. The entire handshake process may also be represented in a state diagram as shown in Figure F-3.

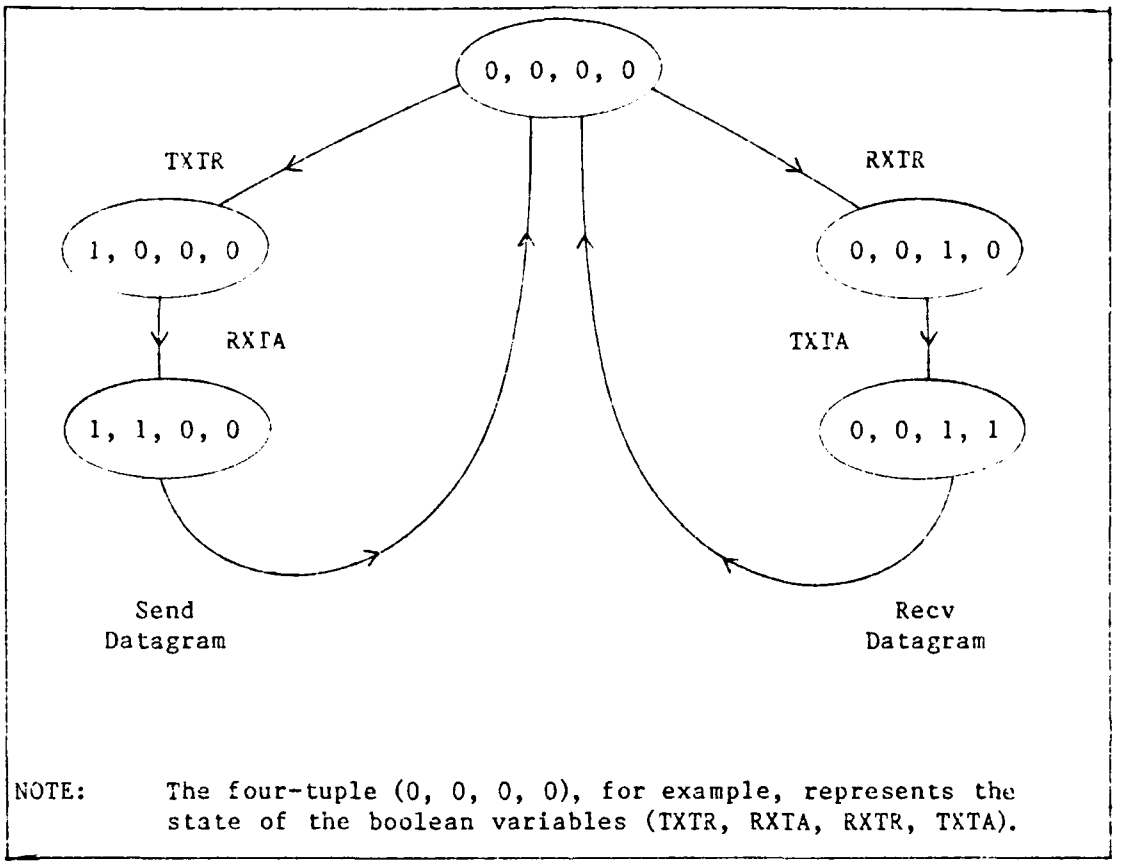


Figure F-3. State Diagram of the TXTR Handshake.

Appendix G

Data Dictionary

This appendix contains the data dictionary for the four programs that comprise the network and data link layer simulations, the validation and test programs used with the UNID II and NETOS, and the host CP/M simulation software (Chapter Five).

The simulation dictionaries are presented first followed by the dictionary for the software on the SBC 544 and the CP/M system. Each of the four programs has its own subdictionary which contains a section for constants, variables, and procedures. Each entry is listed in alphabetical order.

The batch files used to link and locate the object code generated by the compiler are also included at the end of each applicable subdictionary.

The appendix is subdivided into four subdictionaries which are listed as follows:

1. Network Layer Simulation

The purpose of this program is to simulate the network layer software on the Intel Software Development System.

Constants

ASCII(*) - Array of ASCII characters used for converting binary to hex and hex to binary numbers for display on the console.

DATA$GRAM$SIZE - Number of bytes in a datagram (128) received from a host.

DATA$TABLE$SIZE - Number of bytes within a data table.

LRIDEST$ERR - Local route in destination error.

LRODEST$ERR - Local route out destination error.

MAX$COUNTRY$CODE - Maximum number of countries operational on the DELNET.

MAX$NETWORK$CODE - Maximum number of UNIDs operational within a particular country.

PACKET$SIZE - Number of bytes in a packet (133).

PACKETSINTABLE - Number of packets in a packet table.

PACKET$TABLE$SIZE - Number of bytes in a packet table.

R$CONN - I/O handle number for ISIS console call.

STAT$NBR - Number of the status entries to be included in the status table.

SYSMEMBASE - Base address used to locate the shared table and variables.

SYS$BASE - Base label used to properly locate the shared table and variables. Used with SYSMEMBASE.

THIS$COUNTRY$CODE - Unique code indicating in which country THIS$UNID$NBR resides.

THIS$UNID$NBR - Unique UNID number for the UNID performing the interface between local hosts and the DELNET.

TCP$DATA$SIZE - Number of user data bytes in the TCP header.

TA - Transmit acknowledge character.

TR - Transmit request character.

W$CONN - I/O handle number for ISIS console call.

Variables

ACTUAL - Number of characters returned from ISIS console read call.

BUFFER - 128 byte buffer used with ISIS console read call.

DESTINATION - Indicates whether a received datagram is destined for the network or another attached local host.

DESTINATION$ADDRESS - Indicates the destination address of a datagram.

SOURCE$ADDRESS - Indicates the source address of a datagram.

ERRNUM - Number of the error returned from an ISIS system call.

LC01NE - Pointer in the array LC01TB pointing to the next available position for a received datagram.

LC02NE - Pointer in the array LC02TB pointing to the next available position for a received datagram.

LC03NE - Pointer in the array LC03TB pointing to the next available position for a received datagram.

LC04NE - Pointer in the array LC04TB pointing to the next available position for a received datagram.

LC01NS - Pointer in the array LC01TB pointing to the next datagram to service.

LC02NS - Pointer in the array LC02TB pointing to the next datagram to service.

LC03NS - Pointer in the array LC03TB pointing to the next datagram to service.

LC04NS - Pointer in the array LC04TB pointing to the next datagram to service.

LC01SZ - The maximum number of bytes in the LC01TB array.

LC02SZ - The maximum number of bytes in the LC02TB array.

LC03SZ - The maximum number of bytes in the LC03TB array.

LC04SZ - The maximum number of bytes in the LC04TB array.

LC01TB - Local receive table for host port number one.

LC02TB - Local receive table for host port number two.

LC03TB - Local receive table for host port number three.

LC04TB - Local receive table for host port number four.

LPTR$1, LPTR$2, LPTR$3, LPTR$4 - Pointer to the current packet to be passed to the data link layer.

LSEM$1, LSEM$2, LSEM$3, LSEM$4 - Semaphore used by the network and data link layers to indicate the state of the packet transfer.

LSPARE$1, LSPARE$2, LSPARE$3, LSPARE$4 - Spare memory locations used by the SBC 88/45 for it's pointer transfer.

NPTR$1, NPTR$2, NPTR$3, LPTR$4 - Pointer to the current packet to be passed to the network layer.

NSEM$1, NSEM$2, NSEM$3, NSEM$4 - Semaphore used by the network and data link layers to indicate the state of the packet transfer.

NSPARE$1, NSPARE$2, NSPARE$3, NSPARE$4 - Spare memory locations used by the SBC 88/45 for it's pointer transfer.

TX01NE - Pointer in the array TX01TB pointing to the next available position for a transmitted datagram.

TX02NE - Pointer in the array TX02TB pointing to the next available position for a transmitted datagram.

TX03NE - Pointer in the array TX03TB pointing to the next available position for a transmitted datagram.

TX04NE - Pointer in the array TX04TB pointing to the next available position for a transmitted datagram.

TX01NS - Pointer in the array TX01TB pointing to the next datagram to service.

TX02NS - Pointer in the array TX02TB pointing to the next datagram to service.

TX03NS - Pointer in the array TX03TB pointing to the next datagram to service.

TX04NS - Pointer in the array TX04TB pointing to the next datagram to service.

TX01SZ - The maximum number of bytes in the TX01TB array.

TX02SZ - The maximum number of bytes in the TX02TB array.

TX03SZ - The maximum number of bytes in the TX03TB array.

TX04SZ - The maximum number of bytes in the TX04TB array.

TX01TB - Local receive table for host port number one.

TX02TB - Local receive table for host port number one.

TX03TB - Local receive table for host port number one.

TX04TB - Local receive table for host port number one.

MESSAGE(*) - Test message array.

STATUS - Error status of ISIS console I/O calls.

Procedures

DET$ADDR - Determine the destination of the datagram from the attached host.

DET$ADDR$NL - Determine the destination of the datagram passed from the data link layer.

ERROR - I/O error handler for ISIS operating system calls.

EXIT - Graceful method to end the simulation; returns to the ISIS operating system.

INIT - Initializes the variables to their initial states.

INIT$TAB - Initializes the network and data link layer tables and pointers to their initial values.

LDTABHSKP - Housekeep a specified buffer table load pointer.

LOOP - Simulates the semaphore check and set operation of the SBC 88/45 board to turn a frame around to the network layer.

MOVETO$LOCAL - Move a datagram from a receive host buffer or the data link layer buffer to the local host transmit buffer.

READ - Read a line of character input from the console; an ISIS operating system call.

ROUTE$IN - Route received datagrams from the local hosts to the data link layer or the local host transmit buffers.

ROUTE$OUT - Send the datagrams in the transmit buffers to the local hosts.

SEND$PACKET - Transforming the user datagram into a packet for transfer to the data link layer.

SERVICE$LOOP - Turns a frame around at the data link layer. The source and destination headers are exchanged.

SET$TRTA - Queries operator for which host channels will use the TRTA handshake.

SNDSEQ - Takes a message string from the calling procedure and outputs it to the ISIS operating system.

SRVCTABHSKP - Housekeep a specified buffer table service pointer.

WRITE - Write a line of character information to the console; an ISIS operating system call.

Link and Locate Batch File

CAUTION: Do not change address or other parameters in the following batch file. They are highly hardware dependent on the System III and the ISIS operating system.

```
LINK NEWLOC.OBJ,SYSTEM.LIB,PLM80.LIB TO NEWLOC.LNK MAP
LOCATE NEWLOC.LNK TO NEWLOC STACKSIZE(100H) ORDER(CODE,DATA,&
 STACK,MEMORY) CODE(5000H) MAP PRINT(NEWLOC.MP2)
TYPE NEWLOC.MP2
```

2. Data Link Layer Simulation

The purpose of this program is to simulate the data link layer software on the Intel Software Development System.

Constants

ASCII(*) - Array of ASCII characters used for converting binary to hex and hex to binary numbers for display on the console.

CONCTC - Network monitor counter timer port address.

CONCMD - Network monitor USART command port address.

CONDAT - Network monitor USART data port address.

DATA$GRAM$SIZE - Number of bytes in a datagram (128) received from a host.

DATA$TABLE$SIZE - Number of bytes within a data table.

LRIDEST$ERR - Local route in destination error.

LRODEST$ERR - Local route out destination error.

MAX$COUNTRY$CODE - Maximum number of countries operational on the DELNET.

MAX$NETWORK$CODE - Maximum number of UNIDs operational within a particular country.

MAXNOA - Maximum number of timing counts for network channel A.

MAXNOB - Maximum number of timing counts for network channel B.

MAXRETRANS$A - Maximum number of retransmissions of a frame for network channel B.

MAXRETRANS$B - Maximum number of retransmissions of a frame for network channel B.

PACKET$SIZE - Number of bytes in a packet (133).

PACKETSINTABLE - Number of packets in a packet table.

PACKET$TABLE$SIZE - Number of bytes in a packet table.

R$CONN - I/O handle number for ISIS console call.

STAT$NBR - Number of the status entries to be included in the status table.

THIS$COUNTRY$CODE - Unique code indicating in which country THIS$UNID$NBR resides.

THIS$UNID$NBR - Unique UNID number for the UNID performing the interface between local hosts and the DELNET.

TCP$DATA$SIZE - Number of user data bytes in the TCP header.

W$CONN - I/O handle number for ISIS console call.

Variables

ACTUAL - Number of characters returned from ISIS console read call.

BUFFER - 128 byte buffer used with ISIS console read call.

CTCNOA - Progressive number of time counts for network channel A.

CTCNOB - Progressive number of time counts for network channel B.

DESTINATION - Indicates whether a received datagram is destined for the network or another attached local host.

SOURCE$ADDRESS - Indicates the source address of a datagram.

ERRNUM - Number of the error returned from an ISIS system call.

INPUTSEQBIT$A - Input sequence bit number for channal A.

INPUTSEQBIT$B - Input sequence bit number for channel B.

LCNTNE - Pointer in the array LCNTTB pointing to the next available position for a received datagram.

LCNTNS - Pointer in the array LCNTTB pointing to the next datagram to service.

LCNTSZ - The maximum number of bytes in the LCNTTB array.

LCNTTB - Local to network table.

NTLCNE - Pointer in the array NTLCTB pointing to the next available position for a received datagram.

NTO1NE - Pointer in the array NTO1TB pointing to the next available position for a received datagram.

NTO2NE - Pointer in the array NTO2TB pointing to the next available position for a received datagram.

NTLCNS - Pointer in the array NTLCTB pointing to the next datagram to service.

NT01NS - Pointer in the array NT01TB pointing to the next datagram to service.

NT02NS - Pointer in the array NT02TB pointing to the next datagram to service.

NTLCSZ - The maximum number of bytes in the NTLCTB array.

NT01SZ - The maximum number of bytes in the NT01TB array.

NT02SZ - The maximum number of bytes in the NT02TB array.

NTLCTB - Local receive table for host port number two.

NT01TB - Local receive table for host port number three.

NT02TB - Local receive table for host port number four.

MESSAGE(*) - Test message array.

OUTFRAMECHANE - Pointer in the array OUTFRAMECHATB pointing to the next available position for a transmitted datagram.

OUTFRAMECHBNE - Pointer in the array OUTFRAMECHBTB pointing to the next available position for a transmitted datagram.

OUTFRAMECHANS - Pointer in the array OUTFRAMECHATB pointing to the next datagram to service.

OUTFRAMECHBNS - Pointer in the array OUTFRAMECHBTB pointing to the next datagram to service.

OUTFRAMECHASZ - The maximum number of bytes in the OUTFRAMECHATB array.

OUTFRAMECHBSZ - The maximum number of bytes in the OUTFRAMECHBTB array.

OUTFRAMECHATB - Local receive table for host port number one.

OUTFRAMECHBTB - Local receive table for host port number one.

OUTTABFULL - Boolean to determine when the network transmit table contains a frame.

RETRANS$A - Progressive number of retransmissions of a frame for channel A.

RETRANS$B - Progressive number of retransmissions of a frame for channel A.

SEQBITA - Frame acknowledge bit for channel A.

SEQBITB - Frame acknowledge bit for channel B.

STATUS - Error status of ISIS console I/O calls.

THISSEQBIT$A - Current sequence bit for frame to transmit in channel A.

THISSEQBIT$B - Current sequence bit for frame to transmit in channel B.

TIMCHA - Current time count for channel A.

TIMCHB - Current time count for channel B.

Procedures

DET$ADDR - Determine the destination of the packet from the attached host.

DQ$DECODE$EXCEPTION - External ISIS call to decode error exceptions.

DQ$CLOSE - External ISIS call to close an I/O handle.

DQ$DETACH - External ISIS call to detach an I/O device.

DQ$EXIT - External ISIS call to exit the current program back to the ISIS operating system.

DQ$ATTACH - External ISIS call to attach an I/O device.

DQ$CREATE - External ISIS call to obtain an I/O handle.

DQ$OPEN - External ISIS call to open a file.

DQ$READ - External ISIS call to read an opened file.

DQ$WRITE - External ISIS call to write an opened file.

INIT - Initializes the variables to their initial states.

INIT$TAB - Initializes the network and data link layer tables and pointers to their initial values.

LDTABHSKP - Housekeep a specified buffer table load pointer.

LOOP - Simulates the operation of another UNID in the network.

ROUTE$IN - Route received packets and frames from the network layer and the network.

ROUTE$OUT - Send the frames to the network or packets to the network layer.

BUILDIPACKET - Transforms the user packet into a frame for transfer to the network.

SERVICE$LOOP - Turns a frame around in the network. The source and destination headers are exchanged.

SNDSEQ - Takes a message string from the calling procedure and outputs it to the ISIS operating system.

SRVCTABHSKP - Housekeep a specified buffer table service pointer.

Link and Locate Batch File

CAUTION: Do not change address or other parameters in the following batch file. They are highly hardware dependent on the System III architecture and the ISIS operating system.

```
run link86 netnew.obj, small.lib
run loc86 netnew.lnk ad(sm(code(7300h),const(8b00h),data(9900h), &
 stack(bd00h),memory(c300h),??seg(c200h)))
```

3. SBC 544 Validation

The purpose of this program is to operate the network layer software on the Intel SBC 544. All the constants, variables and procedures from the network layer simulation are used in this program with the following exceptions:

1. The READ, WRITE, EXIT and ERROR ISIS system calls are not used.
2. The constants R$CONN and W$CONN are not used.
3. The variables ACTUAL, BUFFER, ERRNUM, and STATUS are not used.

The following constants, variables and procedures are additions to the network layer simulation.

Constants

BRF0, BRF1, BRF2, BRF3 - Data rate factor, USART 0, 1, 2, and 3.

SIM$MASK - Set interrupt mask mask.

MASTER - Port number for Master Mode.

SLAVE - Port number for Slave Mode.

8251A USART Constants:

US$P0$CMD - SERIAL PORT 0 COMMAND
US$P0$STAT - SERIAL PORT 0 STATUS
US$P0$DATA - SERIAL PORT 0 DATA
US$P1$CMD - SERIAL PORT 1 COMMAND
US$P1$STAT - SERIAL PORT 1 STATUS
US$P1$DATA - SERIAL PORT 1 DATA
US$P2$CMD - SERIAL PORT 2 COMMAND
US$P2$STAT - SERIAL PORT 2 STATUS
US$P2$DATA - SERIAL PORT 2 DATA
US$P3$CMD - SERIAL PORT 3 COMMAND
US$P3$STAT - SERIAL PORT 3 STATUS
US$P3$DATA - SERIAL PORT 3 DATA
US$MODE - SERIAL PORT MODE
US$COMMAND - SERIAL PORT COMMAND
US$RESET$CMD - RESET USART
USDTRON - RTS,RXE,DTR,TXE
USCRTCMD - RTS,ER,RXE,DTR,TXE
USTTYCMD - RTS,ER,RXE,TXE
USDTROFF - RTS,RXE,TXE

```
        US$RXRDY - RECIEVER READY
        US$TXE - TRANSMITTER EMPTY
        US$TXRDY - TRANSMITTER READY
        PARITY$MASK - MASK OFF PARITY BIT

    8253 Interval Timer Constants:

        IT1$CONT - INTERVAL TIMER 1 CONTROL
        IT1$CNTR0 - COUNTER 0, USART 0
        IT1$CNTR1 - COUNTER 1, USART 1
        IT1$CNTR2 - COUNTER 2, USART 2
        IT2$CONT - INTERVAL TIMER 2 CONTROL
        IT2$CNTR0 - COUNTER 3, USART 3
        IT2$CNTR1 - COUNTER 4, CNTR5 OR SPLIT CLOCKS
        IT2$CNTR2 - COUNTER 5, RST 7.5
        USART$CNTR$M3 - DIVIDE BY N RATE GENERATOR, MODE 3, FOR USART CLK *
                        16, CLK = 1.2288 MHZ
        B19200 - TIMER VALUE FOR 19.2 KBPS
        B9600 - TIMER VALUE FOR 9600 BPS
        B4800 - TIMER VALUE FOR 4800 BPS
        B2400 - TIMER VALUE FOR 2400 BPS
        B1200 - TIMER VALUE FOR 1200 BPS
        B600 - TIMER VALUE FOR 600 BPS
        B300 - TIMER VALUE FOR 300 BPS
        B150 - TIMER VALUE FOR 150 BPS
        B110 - TIMER VALUE FOR 110 BPS

    8155 Peripheral Interface Constants:

        PI$PORTA - PORT A (OUTPUT)
        PI$PORTB - PORT B (INPUT)
        PI$PORTC - PORT C (INPUT)
        PI$STAT - PPI STATUS
        PI$CMD - PPI COMMAND
        PI$CNTR$LO - PPI COUNTER LO BYTE
        PI$CNTR$HI - PPI COUNTER HI BYTE
        PI$CNTR$LOCNT - PPI COUNTER TIME CONST
        PI$CNTR$HICNT - PPI COUNTER TIME CONST
        PI$INIT$CMD1 - PPI INITIALIZATION COMMAND 1, A OUT, B & C IN, STOP
                       COUNT
        PI$INIT$CMD2 - PPI INITIALIZATION COMMAND 2, A OUT, B & C IN, START
                       COUNT
        PI$INIT$US$INT1 - USART AND INT CONT RESET
        PI$INIT$US$INT2 - USART AND INT CONT NORMAL
        PI$PORTC$STAT - PORT C STATUS
        PI$PORTC$CTL - PORT C CONTROL
        PI$M2M1 - A-MODE 1, B-MODE 2
        PI$OBF - OUTPUT BUFFER READY
        PI$IBF - INPUT BUFFER READY
```

8259 Interrupt Controller Constants:

IC$PORTA - PORT A
IC$PORTB - PORT B
IC$ICW1 - INIT COMMAND WORD 1, (A7A6A5) = 010; EDGE TRIG; INTERVAL = 4; SINGLE; NO ICW4
IC$ICW2 - INIT COMMAND WORD 2, (A15-A0) = 0
IC$ICW3 - INIT COMMAND WORD 3, NO SLAVE IN IR
INIT$MASK - '10101010B', INITIAL INTERRUPT MASK, OCW1; RECEIVE INTR ON, TRANSMIT INTR OFF
IC$EOI - END OF INTERRUPT CMD, OCW2, ROTATE (PRIORITY) ON NON-SPECIFIC EOI
IC$OCW3$SMMS - SPECIAL MASK MODE SET
IC$OCW3$SMMR - SPECIAL MASK MODE RESET

Variables

BYTES$RECV$1, BYTES$RECV$2, BYTES$RECV$3, BYTES$RECV$4 - Integer value indicating how many bytes of a datagram have been received from a host.

BYTES$SENT$1, BYTES$SENT$2, BYTES$SENT$3, BYTES$SENT$4 - Integer value indicating how many bytes of a datagram have been sent to the host.

CHAR$1, CHAR$2, CHAR$3, CHAR$4 - Place holder for the received character in the receive interrupt routine.

RXTA$1, RXTA$2, RXTA$3, RXTA$4 - Boolean flag to indicate if a transmit acknowledge has been received.

RXTR$1, RXTR$2, RXTR$3, RXTR$4 - Boolean flag to indicate if a transmit request has been received.

SEND$1, SEND$2, SEND$3, SEND$4 - Boolean flag to indicate when a host channel is sending data to its host.

TA - Transmit acknowledge character.

TR - Transmit request character.

TRTA$1, TRTA$2, TRTA$3, TRTA$4 - Boolean flags to indicate if the transmit request/transmit acknowledge handshake is in use.

TXTA$1, TXTA$2, TXTA$3, TXTA$4 - Boolean flag to indicate if a transmit acknowledge was sent.

TXTR$1, TXTR$2, TXTR$3, TXTR$4 - Boolean flag to indicate if a transmit request was sent.

Procedures

INITIALIZE$BOARD - Initialize the hardware integrated circuits on the SBC 544.

R$MASK - External procedure to read the interrupt mask on the 8085 processor. Linked from PLM80.LIB.

S$MASK - External procedure to set the interrupt mask on the 8085 processor. Linked from PLM80.LIB.

Link and Locate Batch File

CAUTION: Do not change address or other parameters in the following batch file. They are highly 544 hardware dependent on the SBC 544 architecture and the network layer software.

```
LINK NEWLOC.OBJ,PLM80.LIB TO NEWLOC.LNK MAP
LOCATE NEWLOC.LNK TO NEWLOC STACKSIZE(100H) ORDER(CODE,DATA,&
 STACK,MEMORY) CODE(60H) DATA(0A000H)&
 RESTARTO MAP PRINT(NEWLOC.MP2)
TYPE NEWLOC.MP2
OBJHEX NEWLOC TO NEWLOC.HEX
```

4. Host CP/M Simulation

The purpose of this program is to simulate a host system to the SBC 544 network layer software. All the constants, variables and procedures from the network layer simulation are used in this program with the exception that the console I/O now occurs through the CP/M system calls and the actual character I/O to the UNID II is accomplished through calls to an assembly language module linked to this module.

The following constants, variables and procedures are additions to the network layer simulation.

Constants

ASCII(*) - Array used for converting hex to binary and binary to hex.

BDOS2 - BDOS call 2-console output.

BDOS9 - BDOS call 9-print string until `$'.

BDOS10 - BDOS CALL 10-read console input buffer.

DATA$GRAM$SIZE - Number of bytes from host.

DATA$TABLE$SIZE - Number of bytes in datagram table.

MAX$COUNTRY$CODE - Indicates country codes in use.

MAX$NETWORK$CODE - Indicates UNIDs operational in the network.

MAX$RXTA$TRIES - Maximum number of TA wait tries.

PACKET$TABLE$SIZE - Number of bytes in packet table.

TCP$DATA$SIZE - TCP data size.

THIS$COUNTRY$CODE - Country code where this UNID resides.

THIS$UNID$NBR - Unique address for this UNID in its country code.

Variables

RESULT - Error value returned by BDOS function calls.

BUFFER(128) - Line buffer used for console input.

CHAN$NUM - Channel number in which to load the test datagrams.

DESTNETCODE - Destination network code for the test datagrams.

DEST$HOST$CODE - Destination host code for the test datagrams.

CHAN$PTR - Pointer to the current datagram.

RXTA$TRIES - Number of received transmit acknowledge attempts.

TRANS1RDY - Indicator when the transmit software is ready to transmit a datagram.

RX01NE - Pointer to next available space to receive a datagram.

RX01NS - Pointer to next datagram to service.

RX01SZ - Size of receive datagram buffer.

RX01TB - Receive buffer for datagrams.

TX01NE - Pointer to next available space to send a datagram.

TX01NS - Pointer to next datagram to service

TX01SZ - Size of receive datagram buffer.

TX01TB - Transmit buffer for datagrams.

DESTINATION - Destination of the datagram for program control.

DESTINATION$ADDRESS - Destination address of datagram from IP header.

SOURCE$ADDRESS - Source address of datagram from IP header.

BYTES$RECV - Integer value indicating how many bytes of a datagram have been received from a host.

BYTES$SENT - Integer value indicating how many bytes of a datagram have been sent to the host.

CHAR - Place holder for the received character in the receive interrupt routine.

RXTA - Boolean flag to indicate if a transmit acknowledge has been received.

RXTR - Boolean flag to indicate if a transmit request has been received.

SEND - Boolean flag to indicate when a host channel is sending data to its host.

TA - Transmit acknowledge character.

TR - Transmit request character.

TRTA - Boolean flags to indicate if the transmit request/transmit acknowledge handshake is in use.

TXTA - Boolean flag to indicate if a transmit acknowledge was sent.

TXTR - Boolean flag to indicate if a transmit request was sent.

Procedures

BDOS - External call to the CP/M operating system to perform a BDOS call.

CHK$RXTA - Procedure to check the receive USART for a received transmit acknowledge character.

CHK$RXTR - Procedure to check the receive USART for a received transmit request character.

EXIT - External call to return to the CP/M operating system.

INIT - Procedure to initialize the variables used in the program.

LDTABHSKP - Procedure to adjust the pointers to the next available datagram position in a buffer table.

LOAD - Procedure to interactively load datagrams into a buffer for transmission to the UNID.

LOOP2 - Procedure to send and receive a datagram.

RCV$1 - Procedure to read a datagram from the USART.

READ - Procedure to read a line of buffered input from the host console.

READ$LINE - Procedure to interactively read and interpret a line of text at the host console.

READ$RXTAB - Procedure to read and display the contents of the receive buffer table.

READ$TXTAB - Procedure to read and display the contents of the transmit buffer table.

SCLRCM - External call to clear the USART receive port.

SCMCHK - External call to check the USART receive port for a character.

SCMIN - External call to get a character from the USART.

SCMOUT - External call to send a character to the USART.

SINIT - External call to initialize the host CP/M USART port.

SNDSEQ - Procedure to send a message to the host console for display.

SRVCTABHSKP - Procedure to adjust the pointers to the next to service datagram in the buffer tables.

TRANS$1 - Procedure to send a datagram to the USART.

Link and Locate Batch File

CAUTION: Do not change address or other parameters in the following batch file. They are highly CP/M system dependent.

```
LINK CPMTMP.OBJ,SBS.OBJ,PLM80.LIB TO CPMTMP.LNK MAP
LOCATE CPMTMP.LNK TO CPMTMP STACKSIZE(100H) ORDER(CODE,DATA,&
 STACK,MEMORY) CODE(103H) MAP PRINT(CPMTMP.MP2)
TYPE CPMTMP.MP2
OBJHEX CPMTMP TO CPMTMP.HEX
```

VITA

Creed T. Childress, Jr., was born on 7 October 1943 in Santa Barbara, California. He graduated from South Lake Tahoe High School in 1961. He then attended the University of California at Berkeley, California State University at Sacramento, and the University of California at Santa Barbara before enlisting in the U.S. Air Force in October 1966 when he began training as a Radio Relay Maintenance Technician. After his initial training, he was assigned to the 2063 Communications Group at Lindsey Air Station, Wiesbaden, Germany, where in 1972, he was accepted into the Airmen Education and Commissioning Program. He attended the University of Michigan, Ann Arbor, Michigan, and graduated in May 1974 with a Bachelor of Science in Industrial and Operations Engineering and a Bachelor of Science in Electrical Engineering. He then attended Officer Training School and was commissioned in August 1974 and assigned to Keesler Technical Training Center, Keesler AFB, Mississippi, for further training in the Communications Engineer specialty. His next assignment began in June 1975 as an instructor in the Wideband Systems Evaluation School (AFCC) at Richards-Gebaur AFB, Grandview, Missouri. In November 1977, he was assigned as the Detachment Commander of Detachment 27 of the 2187 Communications Group in Martina Franca, Italy. Following that challenging assignment, he was assigned to the Plans Division of the Headquarters of the Air Force Operational Test and Evaluation Center, Kirtland AFB, New Mexico in December 1979. He entered the Air Force Institute of Technology in May 1983.

Permanent Address: 9121 Cecile Way
Sacramento, CA, 95826

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A151935

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/84D-17	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/EN	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright Patterson AFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.; PROJECT NO.; TASK NO.; WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): See Box 19	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM ___ TO ___	14. DATE OF REPORT (Yr., Mo., Day): 1984 December; 15. PAGE COUNT: 331
16. SUPPLEMENTARY NOTATION: bookbp	

17. COSATI CODES		
FIELD	GROUP	SUB GR
09	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Local Area Networks, Network Interface, Protocols, UNID, ISO Reference Model, X.25, X.121, TCP/IP

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II IN THE DIGITAL ENGINEERING LABORATORY NETWORK (DELNET)

Thesis Advisor: Dr. Gary Lamont

Author: Creed T. Childress, Jr., BS EE, BS IOE, Capt USAF

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr Gary Lamont	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3576; 22c. OFFICE SYMBOL: AFIT/EN

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

Block 19 (cont)

Abstract

This research effort describes the continued development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two main hardware modules; a local module for the network layer software and a network module for the datalink layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an iSBC 544 and the network module is an SBC 88/45. The network layer software supports the CCITT X.25 protocol, datagram option, and the data link layer software supports the CCITT X.25 LAPB (HDLC) protocol. This report documents the detailed hardware and software design, integration, validation and test of this system.

END

FILMED

5-85

DTIC